Overview of Tiger Server - Part II

Learn To Draw With Quartz 2D

MacTech Magazine
August • 2005

MACTECH®

The Journal of Macintosh Technology

Hot New Tiger Info!!

Building Automator Actions With AppleScript

By Benjamin S. Waldie

This Month's Getting Started:
PHP, MySQL and Forms

Find Out Why
The Source Hound
is Nuts About SquirrelMail!

Mac In The Shell:
DNS and E-mail
Troubleshooting - Part II

From The Security Beat:
Securing E-mail with GPG

Palm On OS X:
Dissecting The PDB File

$8.95 US
$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.

08
0 32128 74887 8

Need to find something fast?

INDEX INDEX INDEX

With c-tree Speed.

FairCom's c-tree Plus®

embedded database engine offers Superior Indexing Technology – the key to performance, data integrity, and concurrency. c-tree Plus offers direct record-oriented C and C++ APIs with an industry-standard SQL interface that allows use of any combination of APIs within the same application. Furthermore, we offer source code access for intimate programming control, unmatched portability, and developer-to-developer technical support.

Heterogeneous Environments

c-tree Plus and c-treeSQL™ Servers are the perfect solution for your mixed platform environments. Mac servers to Windows clients? No problem. Linux Servers to Mac clients? No problem. c-tree has a long history of cross-platform development solutions. Byte incompatibility between platforms is handled seamlessly with our Uniformat data handling technology.

Low TCO

c-tree is priced affordably, requires minimal hardware resources, and needs no IT staff for maintenance. If Total Cost of Ownership (TCO) is important to you, c-tree is the perfect database.

Easy Deployment

c-tree Servers are designed for ease of use and deployment as well. Out of the box, our servers can be installed and running in minutes.

Start indexing your data today!

TRY IT NOW!

FairCom®

USA • Europe • Japan • Brazil

www.faircom.com

Go to www.faircom.com/go/mteval for a FREE evaluation of c-tree Plus!

Letter From The Editor

Ah, August - Summer At It's Best!

We don't know about you, but for those of us at MacTech, (the magazine has it's offices in Los Angeles and Arlington, VA) it's been hot this summer. Very hot! But that's the way we like our weather, and it's the way we like the marketplace we are in, the Mac technology market!!

To say Apple's been hot would be a huge understatement! In its fiscal 3rd Quarter, ending June 25th, Apple reported the highest revenue and profit in its history. It's now a $14billion company. Apple shipped almost 1.2 million Macs in the quarter and over 6 million iPods. (Know anyone who doesn't have one? Other than your parents?) It's an exciting market again, and we couldn't be more thrilled to be an integral part of it!!

As good as Apple's financial results are, the technology side of things is just as strong. We think Tiger is clearly the best commercial OS that has ever been released. The opportunities for Apple to make significant inroads in the Enterprise market are enormous. In our heart of hearts we hope Apple seizes on this opportunity and shouts its very strong Enterprise message from every mountaintop and in every corporate CIO's office around the world. We sure are doing our part to help. No problem Steve, don't mention it! ☺

We have another great line-up of articles and columns for you this month. While last month we learned Dave Mark has become a boater, (here's fair warning to all you boaters, swimmers and Jet Skiers on Virginia's beautiful Lake Anna!!) this month Dave continues his journey into PHP/MySQL with a great Getting Started on implementing HTML forms. For those

MACTECH®
A publication of
XPLAIN CORPORATION

TABLE OF CONTENTS

DEPARTMENTS

FEATURED ARTICLES

Letter From The Editor continued...

of you who have been hanging on Dave's every word in this series, you are going to really enjoy and learn from this great column. Dave also makes mention of a really wonderful new book from Mac legend Scott Knaster. The book is called *Hacking Mac OS X Tiger: Serious Hacks, Mods and Customizations*. You can find more about this wonderful book, that we very highly recommend, and can even read an excerpt chapter at:

http://www.wiley.com/WileyCDA/WileyTitle/productCd-076458345X.html

Our featured cover story this month is from our AppleScript Editor, Ben Waldie. From the feedback we received we know all of you have been reading Ben's great AppleScript columns each month. This month Ben introduces us to the exciting world of creating Automator Actions with AppleScript. For anyone who is comfortable in AppleScript, and wants to learn more about Automator, this column is a must!

In this month's installment of Mac In The Shell, Ed Marczak brings us Part II of his exploration of DNS and E-mail. After a fine introductory piece last month on the basics, this month Ed shows us how to use the Terminal to really dig-in, isolate the sources of trouble and resolve issues quickly. This piece is a must for anyone who is administering mail services and wants to use the Terminal as a primary tool to facilitate this task.

For us, one of the best parts of attending WWDC is the opportunity to meet exciting new talent in the Mac technical community. While on the bus ride from Moscone to Cupertino for this year's big Apple bash, (anyone else like the Wallflowers?), we met and got to know a consultant and writer we are excited to be bringing to our readers. His name is Emmanuel Stein and he runs a consulting company in NYC called Macverse. In his debut for MacTech, Emmanuel is checking in from the Security Beat with a piece on using GPG to secure your mail. Read it and let us (and Emmanuel at macverse@mac.com) know what you think. In the coming months be looking for pieces from Emmanuel on demystifying the data storage alphabet soup and Open Source package management tools like Pink and Open Darwin Ports.

We have another new contributor with us this month. Jose Cruz, a freelance engineering consultant

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Letter From The Editor continued...

from British Columbia, brings us a really great, in-depth piece on progaming for the Palm OS on OS X. The Piece, *Dissecting The PDB File*, is a very strong exploration of using Cocoa, Core Foundation and XCode to build applications and utilities for the Palm OS. We really enjoy Jose's writing and will be running more pieces from him in future issues of MacTech.

After a few months away, frequent contributor Paul Ammann is back with us this month. Check out his really fine article on Exclusivity and file access in OS X, *Exclusive File Access in Mac OS X*. David Hill is also back this month with a really enjoyable and informative piece on OS X's 2D drawing API, Quartz 2D, *Mineralogy 101*. Enjoy!

After a month's hiatus (either it's hot in Chicago too or his ten rounds with Michael Bell just wore him out!) our resident Open Source bigot Dean Shavit, introduces us to the wonderful world of SquirrelMail and shows us how to get the most out of this really super open source WebMail package. Welcome back Source Hound! We missed ya!!

Tim Monroe, our resident QuickTime guru, this month introduces us to the wonderful world of Threads and, more specifically, performing QuickTime operations of them. We really love this current series from Tim and we are sure you will enjoy the latest installment of the MacTech QuickTime Toolkit! Be reading next month when Tim explores all the hottest stuff introduced in QuickTime 7!

John Welch brings us Part II of last month's cover story on Tiger Server. Picking up where he left off last month, John focuses on Workgroup manager and all the tools Tiger Server provides to help us better manage networks. Read John's piece to learn how to set-up sharepoints and access policies for those sharepoints, set client machines and client machine groups, their access policies and preferences, set users and user groups, their access policies and preferences and perform manual manipulation of Open Directory data.

All this and we get to live in America too! What could be better than this? We're hungry for your feedback. Let us know what you think at editorial@mactech.com.

Until next month, we hope you enjoy another great issue of MacTech!!

MACTECH

Communicate With Us

DEPARTMENT E-Mails

Orders, Circulation, & Customer Service
cust_service@mactech.com

Press Releases
press_releases@mactech.com

Ad Sales
adsales@mactech.com

Editorial
editorial@mactech.com

Online Support
online@mactech.com

Accounting
accounting@mactech.com

Marketing
marketing@mactech.com

General
info@mactech.com

Web Site
http://www.mactech.com

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact MacTech Magazine Customer Service at 877-MACTECH

We love to hear from you! Please feel free to contact us with any suggestions or questions at any time.

Write to letters@mactech.com or editorial@mactech.com as appropriate.

Data Banks Communications Inc

HOME OF THE FREE COMPUTER
THAT'S RIGHT FREE COMPUTER!

GET A MAC MINI OR DELL COMPUTER FREE JUST SIGN UP FOR VOIP DSL BUNDLE

INFOSPEED 768K/128K	$ 69.95
INFOSPEED 1.5M/128K	$ 69.95
INFOSPEED 1.5M/384K	$ 69.95
INFOSPEED 7.1M/768K	$ 199.95
INFOSPEED 384K/384K	$ 69.95
INFOSPEED 768K/768K	$ 149.95

TeleSoho 1.5M/128	$69.95
TeleSurfer Plus 608K/128K	$69.95
Residential 768K/128K	$69.95
SOHO1.5M/768K	$79.95
SOHO3.0M/384K	$79.95
SOHO3.0M/768K	$79.95
SOHO6.0M/768K	$139.95
TeleSpeed 144K/144K	$125.95
TeleSpeed 192K/192K	$125.95
TeleSpeed 384K/384K	$149.95
TeleSpeed 768K/768K	$159.95
TeleSpeed 1.1M/1.1M	$199.95
TeleSpeed 1.5M/1.5M	$249.95

Ask for business line special!

www.databanksglobal.com

For questions regarding sales or service, please call us 24 hours a day at 1-866-624-6114

PHP, MySQL, And Forms

In our last PHP/MySQL exploration, we used the MySQL monitor to build a database, create a new table, then populate the table with data. We *did* use our PHP chops to query the table and spit out some HTML with the results, but that's a pretty bare-bones/hard-coded approach. A more common technique is to use an HTML form to populate the table. For example, suppose you were building an order-entry system for your sports memorabilia shop. You could use the MySQL monitor to enter all the data that represents your inventory. If you are really comfortable with the monitor, that's not a terrible approach. But, what if you want your assistant, who is a whiz at the web but has no MySQL experience, to manage the data? Creating a series of HTML-driven forms is definitely the way to go.

Before we dig into this month's sample code, I'd like to take a momentary detour to talk about a terrific new book I just got.

Hacking Mac OS X Tiger

Back in the early days of Macintosh, there were just a very few Mac programming titles. One of the best of these books was called ***Macintosh Programming Secrets***. Written by Scott Knaster and Keith Rollin, the book quickly became a cult classic. Well, Scott is back and his latest book is every bit as cool as ***Macintosh Programming Secrets***. The long title is ***Hacking Mac OS X Tiger: Serious Hacks, Mods, and Customizations***. This is no idle boast. The book is chock full of wonderful insider info, and is just plain fun.

The book starts off with a collection of tips. For example, try holding down the option key, then pressing the volume up (or down) key on your keyboard. Cool! The Volume System Pref opens. There are tips for the Finder, the Dock, Dashboard, System Prefs, iTunes, and a lot more. To me, the book is worth the price of admission just for that part along. But wait, there's more!

The second part of the book is called Mods, and explores more developer-oriented things. Things like Automator, Xcode, Property Lists, and Application Bundles. Lots of great info here, especially if you are relatively new to Mac development.

The third part of the book might be my favorite. It's called Hacks, and is full of, well, um, hacks! You'll customize dock icon behavior, hack some Dashboard

widgets, even redirect your web cam output to create a live video desktop.

All this stuff is presented in Scott Knaster's witty, irreverent style. I totally love this book! ***Hacking Mac OS X Tiger*** is part of Wiley's ***ExtremeTech*** series. You can find it at:

http://www.wiley.com/WileyCDA/WileyTitle/productCd-076458345X.html

And now back to our regularly scheduled programming...

Implementing a Form with PHP

The key to implementing an HTML form is the <form> tag. Typically, the <form> tag will include both action and method attributes. The action attribute specifies the name of the file to which to send the form's results. The method attribute specifies how form data is passed along to the action recipient and is either GET or POST. The primary difference between GET and POST is that GET causes the form data to be embedded in the action's URL, while POST passes the data to the action as an input stream (via stdin, for example). GET forms can be bookmarked, since the data is embedded in the URL. POST forms cannot be bookmarked.

In general, I use POST as my default unless I specifically need a URL to be bookmarkable. You've undoubtedly clicked on your browser's reload button and been asked, "Do you really want to reload this form?" or something similar. The page in question is a POST request.

Good to know the difference between GET and POST, use whichever works best in any situation. Note that GET is the default. Note also that if your form contains passwords and the like, the GET URL will include the password data. Not a great idea if you will be passing the URL around.

Embedded between the <form> and </form> tags is the form content. The form content is made up of HTML formatted text, along with a variety of <input> tags. Each <input> tag represents an HTML form element, such as a popup menu, text field, non-echoing text field (for entering passwords), checkbox, radio button, submit button (a pushbutton whose purpose is to submit the form) or a reset button (a pushbutton used to clear the form).

An example will make this a bit clearer.

A PHP Form Example

This example puts up a pair of text fields, with appropriate labels. One field is a standard text field, the second a non-echoing text field. We'll use the former to enter your name and the latter to enter a password. We'll end the form with a submit button. When the submit button is pressed, the same page will be reloaded. The technique of taking two different actions on the same page of PHP code is a common PHP theme. While this is not a particularly useful example, it demonstrates this concept nicely.

Using BBEdit, or your favorite plain-text editor, create a new plain-text file and save it as ***php_form01.php***, then type the following source code into the file:

```
<html>
  <head>
    <title>Sample PHP Form</title>
  </head>
  <body>
    <?php
      if ( !empty( $_POST[ 'name' ] ) ) {
        echo "Your name is {$_POST[ 'name' ]}.<br />";
        echo "Your password is {$_POST[ 'pwd' ]}.<br />";
      }
    ?>
    <form action="<?php $PHP_SELF; ?>" method="POST">
      Name:
      <input type="text" name="name" />
      <br />
      Password:
      <input type="password" name="pwd" />
      <br />
      <input type="submit" />
    </form>
  </body>
</html>
```

Save the file and copy it into the ***Sites*** directory in your home directory. For example, I placed my copy of this code into */Users/davemark/Sites/*.

Once this is done, fire up your favorite browser and enter this URL:

http://127.0.0.1/~davemark/php_form01.php

Be sure to replace davemark with your own user name. Also, be sure to keep the tilda (~).

Once you enter the URL, you should see results similar to those shown in Figure 1. If you don't, be sure your PHP server is running. To do this, create a new, plain-text file named ***test.php***, fill it with these 3 lines of code:

```
<?php
phpinfo()
?>
```

Drag the test file into your ***Sites*** directory and type this URL into your browser:

http://127.0.0.1/~davemark/test.php

Remember to replace davemark with your user name. If the large PHP table appears in your browser, PHP is running and you've got an error in your version of ***php_form01.php***. If the large PHP table does ***not*** appear, go back to the installation instructions for PHP and make sure your server is up and running.

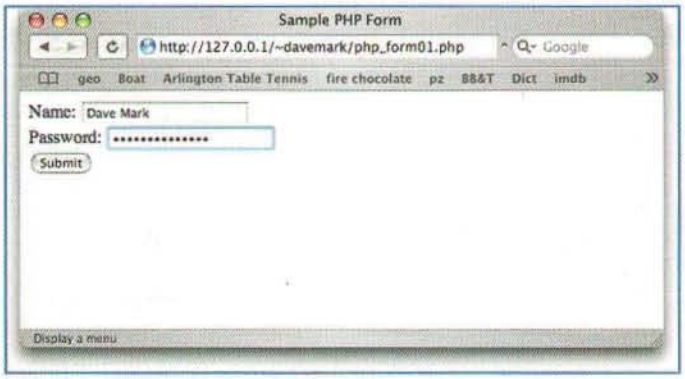


Figure 1. The Sample PHP Form in action.

Click in the *Name* field and type your name. Next, tab over to or click in the *Password* field and type a password. Note that the password characters echo as dots. I'll show you why in a second.

Finally, click the Submit button. You should see something like the output shown in Figure 2. Notice that the name and password are both echoed and the form is then reloaded so you can try this again.

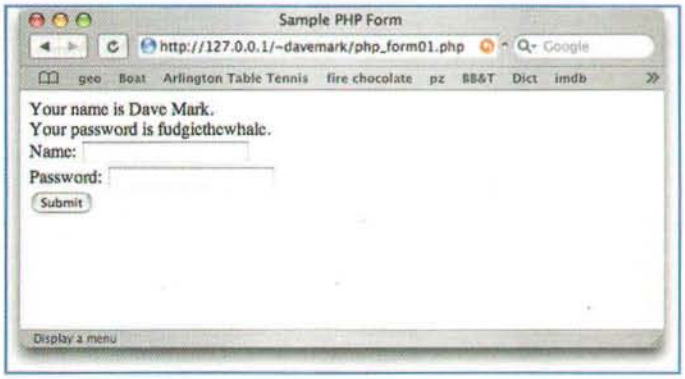


Figure 2. The name and password are echoed and the form is reloaded.

Let's take a look at the code, see how this works.

We start with the basic <html>, <head>, <title>, and <body> tags. Nothing new there.

```
<html>
  <head>
    <title>Sample PHP Form</title>
  </head>
  <body>
```

The first thing we do in the HTML body is open a PHP tag and jump into an if statement. The if statement calls the PHP function ***empty()*** to see if the variable ***$_POST['name']*** is empty, that is, see if it has been set to a value.

Whenever you see a PHP function and don't know what it is, make your way over to http://php.net, type the function name in the ***search for*** text field, select ***function list*** from the ***in the*** popup menu, then click the arrow to do your search. The result of my search for the ***empty*** function is shown in Figure 3.

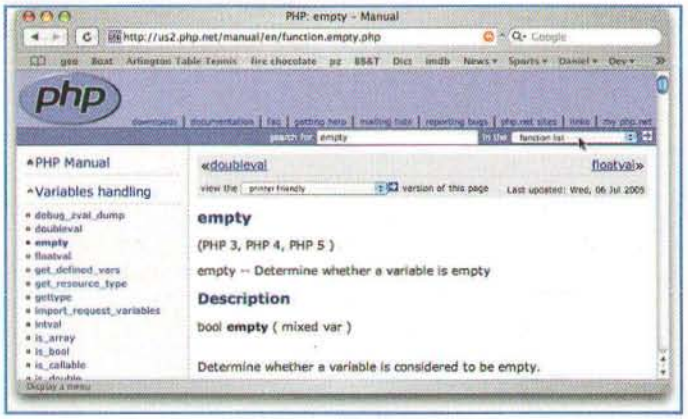


Figure 3. Searching for the empty function on php.net.

What the heck is ***$_POST['name']***? What a weird name for a variable. The variable ***$_POST*** is an ***associative array***. An associative array is an array that is indexed by name instead of by numerical index. For example, you might have an array of nicknames for all your friends, where $nicknames['Fred'] is set to the string 'Stubby', while $nicknames['Phil'] is set to 'Scooter'.

Note that the quotes are optional for single word strings. So you might see $nicknames[Fred]. Though that will work, leaving out the quotes will make your code a bit more cryptic, so use them unless you find yourself in a funky situation where they just get in the way (quoted string inside another quoted string, for example). And even then, be sure to comment your code so everyone can follow along.

$_POST is an associative array that contains an element for each of your form's input fields. So $_POST['name'] contains the contents of the input element with the name "name". $_POST['pwd'] contains the contents of the input element with the name "pwd". You get the idea.

The goal of the if statement is to see if the name field is empty. If not, we'll use echo to display the contents of the name field. Want to learn about echo? Go to php.net and type echo in the search field, search the function list. About ? of the way down the page, you'll see a nice comment about using braces. This is worth reading. In a nutshell, you can use braces as separators when a space character just won't do. For example, in the code below, we used the braces to set off the name $_POST['name'] without putting a space between the end of the variable and the period. This lets us put a period immediately after your name.

Note that we included the
 HTML tag in our output. Remember, the PHP code spits out HTML as its output and then the HTML is passed back to the browser. The
 tag is used to put a return after the name and again after the password.

```
<?php
    if ( !empty( $_POST[ 'name' ] ) ) {
      echo "Your name is {$_POST[ 'name' ]}.<br />";
      echo "Your password is {$_POST[ 'pwd' ]}.<br />";
    }
  ?>
```

If the name field was filled, we'll already have printed the name and password before we get to this point. If the name field was empty, nothing will have been generated yet. Either way, we are now going to put up the form.

The <form> tag takes an action attribute and a method. We already know about the method. The action is the file you want served up with the results of your action. We could have created a second page that displayed the results of our form. But this self-referential approach is pretty common in PHP. We used the variable $PHP_SELF to have the form pass the results on to the same file, sort of like a function calling itself. Want to learn more about $PHP_SELF? It's not a function, so we'll need to search the whole site instead of just the function list. When you search all of php.net, you'll get a Google list of pages to look at. Here's a good place to start:

http://us3.php.net/reserved.variables

You'll need to search the page for PHP_SELF. Note that the $ is not part of the variable name.

```
<form action="<?php $PHP_SELF; ?>" method="POST">
      Name:
```

The two <input> tags specify a text field and a password field. Both do the same thing, but the password field echoes input as dots instead of the actual characters. Note that the name attributes of each <input> tag are what link the results to the $_POST associate array index.

```
  <input type="text" name="name" />
      <br />
      Password:
      <input type="password" name="pwd" />
      <br />
```

Finally, the last bit of <input> for the form is the submit button.

```
  <input type="submit" />
    </form>
  </body>
</html>
```

Type your name and password again and again. Each time you click submit, the page reloads, draws your name and password, and then redraws the blank form. Cool!

Adding MySQL to the Picture

Our next step is to add MySQL to the picture. We'll build a database, using the MySQL monitor, create a new table, then use the form we created to populate the database. If you are new to the MySQL monitor, go back June's ***Getting Started*** column for a review.

Fire up Terminal and start the MySQL monitor. As a reminder, I set up an alias for mysql:

```
alias mysql='/usr/local/mysql/bin/mysql'
```

Here's my sequence for starting MySQL, building the database, and building the table:

```
Daves-Computer:~ davemark$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16 to server version: 4.1.12-
standard

Type 'help;' or '\h' for help. Type '\c' to clear the
buffer.

mysql> create database myaccounts;
Query OK, 1 row affected (0.40 sec)

mysql> use myaccounts;
Database changed
mysql> create table passwords(
    -> name varchar(60),
    -> pwd varchar(60),
    -> userid int(10) auto_increment primary key );
Query OK, 0 rows affected (0.08 sec)

mysql> select * from passwords;
Empty set (0.09 sec)
```

In the above, I first created a database called ***myaccounts***, set myaccounts as the current database, then created a table within myaccounts called ***passwords***. Passwords contains a 60 character name, a 60 character password, and an auto-incrementing userid. We then verified that the passwords table was empty.

Let's switch back over to PHP and modify our code so we are writing our passwords into the table. Back in your plain-text editor, replace your existing code with this version:

```
<html>
  <head>
    <title>Sample PHP Form</title>
  </head>
  <body>
    <p>Connecting to the database...</p>
    <?php
      $host = 'localhost';
      $user = 'root';
      $pw = '';
      $db = 'myaccounts';

      $link = mysql_connect( $host, $user, $pw )
        or die( 'Could not connect: ' . mysql_error() );
      echo 'Connected successfully';
      mysql_select_db( $db );
    ?>
    <p>If we got here, we've connected!</p>

    <?php
      if ( !empty( $_POST[ 'name' ] ) ) {
        $tempname = $_POST[ 'name' ];
        $temppass = $_POST[ 'pwd' ];
        $query = "insert into passwords values (
'$tempname', '$temppass', 0)";
        $results = mysql_query( $query )
        or printf( "Query error: %s", mysql_error() );
      }
      mysql_close();
    ?>
    <form action="<?php $PHP_SELF; ?>" method="POST">
      Name:
      <input type="text" name="name" />
      <br />
      Password:
      <input type="password" name="pwd" />
      <br />
      <input type="submit" />
    </form>
  </body>
</html>
```

The first chunk of this code should be familiar from earlier columns. We use ***mysql_connect()*** to connect to

the database and *mysql_select_db()* to make the database the current one for subsequent commands (just like *use myaccounts;* in the MySQL monitor).

```
<html>
  <head>
    <title>Sample PHP Form</title>
  </head>
  <body>
    <p>Connecting to the database...</p>
    <?php
      $host = 'localhost';
      $user = 'root';
      $pw = '';
      $db = 'myaccounts';

      $link = mysql_connect( $host, $user, $pw )
        or die( 'Could not connect: ' . mysql_error() );
      echo 'Connected successfully';
      mysql_select_db( $db );
    ?>
    <p>If we got here, we've connected!</p>
```

The next chunk of code looks like what we did earlier in the column. Instead of using *echo* to print out the form fields, we build a query instead and use *mysql_query()* to fire it off. This will insert the row into the MySQL table. It is important to note that the *$query* line is a single line of code. It wraps in this column, but you should enter it as a single unbroken quoted string in your source code.

After the if statement, we close the database. Each time this page is loaded, the data base is reconnected, worked with, then closed again. Some people put the *mysql_close()* at the very bottom of the file, just to make sure they don't accidentally place code beneath it.

```
    <?php
        if ( !empty( $_POST[ 'name' ] ) ) {
          $tempname = $_POST[ 'name' ];
          $temppass = $_POST[ 'pwd' ];
          $query = "insert into passwords values (
'$tempname', '$temppass', 0)";
          $results = mysql_query( $query )
          or printf( "Query error: %s", mysql_error() );
        }
        mysql_close();
    ?>
```

The rest of the code is pretty much the same as what you had above, the implementation of the form and the close body and html tags.

```
<form action="<?php $PHP_SELF; ?>" method="POST">
      Name:
      <input type="text" name="name" />
      <br />
      Password:
      <input type="password" name="pwd" />
      <br />
      <input type="submit" />
    </form>
  </body>
</html>
```

Running the MySQL Form

Let's take this puppy for a spin. First, be sure you saved your source code and copy the file to your *Sites* folder. In your browser, type this link:

http://127.0.0.1/~davemark/php_form02.php

Note that I saved this new version of the source code as *php_form02.php*. Be sure to change *davemark* to your user name.

Once the table loads, enter a name and a password (see Figure 4) and click the *Submit* button. Note that each time the page reloads, the database is reconnected.

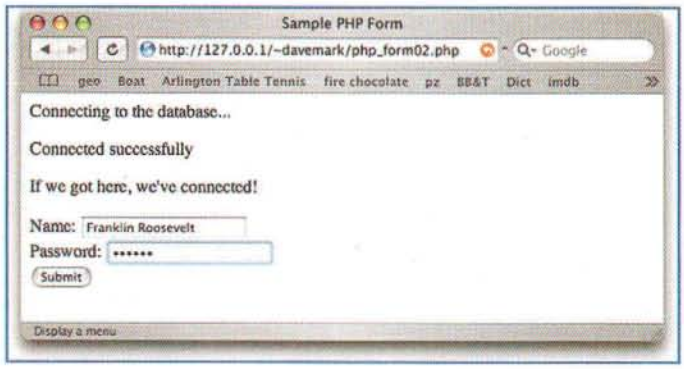


Figure 4. Entering the data that will be written to the MySQL table.

Once you've entered enough data, go back to Terminal and take a look at your table.

```
mysql> select * from passwords;
+-----------------+-------------+--------+
| name            | pwd         | userid |
+-----------------+-------------+--------+
| Franklin Roosevelt | Delano      |      1 |
| Dave Mark       | fudgiethewhale |      2 |
| Dark Helmet     | spaceballs  |      3 |
+-----------------+-------------+--------+
3 rows in set (0.00 sec)
```

Cool! All the data is there.

Until Next Month...

There is just a *ton* you can do with MySQL and PHP. If you don't have a PHP library already, I found the books *MySQL in a Nutshell* and *Programming PHP* from O'Reilly and *PHP 5 and MySQL* from Apress a big help in getting started.

Not sure what I'm going to do for next month's column. I've been reading the galleys to Mark Dalrymple and Scott Knaster's upcoming *Learn Objective-C* from Spiderworks. Hmmm, might be fun to explore a bit of that. Well, we'll see. For now, it's back to Lake Anna for a bit more summer laziness. See you next month. ☺

About The Author

Dave Mark is a long-time Mac developer and author and has written a number of books on Macintosh development. Dave has been writing for MacTech since its birth! Be sure to check out the new Learn C on the Macintosh, Mac OS X Edition at http://www.spiderworks.com.

Shine.

Apache Bootcamp

Cocoa® Bootcamp

Core Bootcamp

PHP 5 Bootcamp

PostgreSQL Bootcamp

Python® Bootcamp

Core Mac OS® X Bootcamp

Get your dark sunglasses ready. Tell your boss to roll out the red carpet. Are you ready for your breakthrough role? Director Mark Dalrymple, author of *Core Mac OS® X and Unix Programming*, will take you from indies to a blockbuster in five days.

Sample Scenes:

- Multithreading
- Rendezvous™
- System Daemons
- Distributed Objects
- Network Programming
- Authentication & Authorization

Big Nerd Ranch offers intensive training classes for developers and administrators. Your expert instructor guides you through a rigorous week of learning. You leave ready to start developing (but with instructor support if you get stuck).

Big Nerd Ranch: Baby, we'll make you a star!

www.bignerdranch.com • 678.595.6773 • roundup@bignerdranch.com

Security Beat

Securing Mail With GPG

A Graphical Tutorial

By Emmanuel Stein

Why Use Encryption at All?

Some people think that if they are not doing anything wrong or illegal, they do not need to encrypt their messages. The problem with this argument is that it assumes that one is trying to protect a message from some government authority or law enforcement apparatus and ignores the very real risks associated with theft of intellectual property via corporate espionage. The fact is, that almost anyone with a bit of networking knowledge can easily "sniff" your message off the network and not only read it, but re-write it and send it along to the intended recipient without you ever being aware (this is called the "man in the middle attack"). Using GPG, you can thwart these would-be crackers and do so transparently by using GPG to encrypt your mail messages.

GPG in a Nutshell:

GPG employs public key, as well as symmetric key cryptography, to perform its magic. When setting up GPG you have several options for generating your keys. The default option uses DSA to generate your primary keypair for signing, and then generates a second (also called subordinate) keypair that uses ElGamal for encryption.

Although we are using the more sophisticated primary and subordinate designations to explain what gets generated in GPG, both the primary and subordinate keychains are usually, for practical purposes, thought of as a single keychain. In this scheme, you have a public and a private key. Your private key is secret and should be treated as such. Take precautions, and do not lose it as you will have to regenerate the key and will not be able to read mail encrypted with your original public key. You do, however, have the ability to revoke a key that is compromised. Unlike your private key, your public key should be widely distributed, allowing others to send you an encrypted message that only you can decrypt with your private key.

GPG uses the same approach to key management as PGP, which is to say via a "web of trust." Rather than employing a centralized certification authority such as Verisign, GPG allows individual users to act as their own certification authority. There is an infrastructure in place for the GPG community of well-maintained keyservers that facilitate exchange of public keys among users.

GPG and Apple Mail: Two Great Tastes That Go Great Together!

For this tutorial, we will focus on enabling GPG in Apple Mail 2.0 using the GPGMail plug-in. Links to sites where one can obtain GPG mail extensions for other popular mail clients will also be included.

How GPG-Enabled Mail Works?

Before we dive into setting up mail for GPG, I will take a moment to review how GPG works in the context of email communications. When you want to send an encrypted message to a colleague or friend, you encrypt the message using their public key, which may be obtained directly from that individual or via a keyserver. The recipient then will decrypt your message using their private key. Signing is also an important part of GPG-enabled mail communications, and involves using your private key to create a digital signature that guarantees to your recipient that the message came from you and has not been tampered with.

GPG Install How-To

What you need:

Go to http://macgpg.sourceforge.net/ and download the following required applications:

- Gnu Privacy Guard (aka GPG)
- GPG Keychain Access

We will not be covering the excellent suite of file utilities also available at this site. Nonetheless, you are encouraged to explore these binaries at your leisure, as they may prove extremely useful for non-mail GPG operations.

In order to enable GPG in Apple Mail, you will also need to obtain the GPGMail application at http://www.sente.ch/software/GPGMail/English.lproj/GPGMail.html.

If you are a security conscious person, and are deploying this package for a mission critical application, then you may want to check each packages authenticity via its MD5 checksum using the openssl command in the terminal.

Installing Gnu Privacy Guard

Double click on the GnuPG .dmg file in order to mount the contents of the archive. To install GPG, double click on the GPG .mpkg file to invoke the installer.

Once GPG has installed successfully, we will need to generate our keypair. To simplify the process, we will use the GPG Keychain Access application rather than the terminal. This application is modeled after the intuitive Keychain application in OS X and is very easy to use. Simply unzip the folder called GPG Keychain Access and drag it into the Applications folder.

To generate your keypair, open the GPG Keychain Access application. The application will notify you that you do not yet have a keypair and offers you the option to generate one (Figure 1). Also, this dialog gives you the option to import your secret key. This is useful when you want to configure a second computer with GPG. Your private or secret key will be created in your home folder under the hidden .gnupg directory.

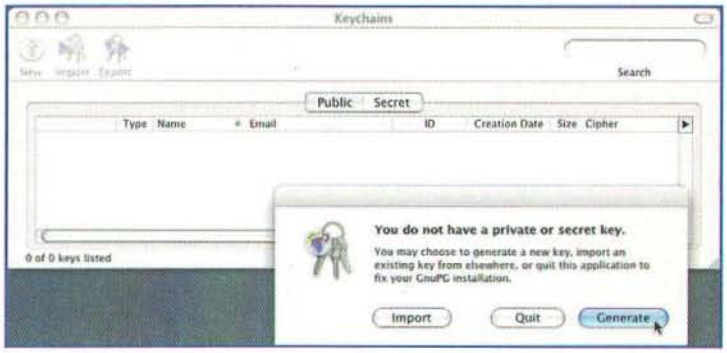


Figure 1. Keypair Generation and Import Dialog

To create a new keypair, select the Generate option in the dialog (Figure 1) to invoke the install wizard. You will be given several options for generating a key (Figure 2). It is recommended that you select the default option, DSA and ElGamal as it offers the most flexibility and is required for use with mail since the other options only allow for signing and not encrypting.

Figure 2. Key Generation Wizard: Algorithm Selection

Once you have selected your algorithms (basically the functions used to scramble and de-scramble your data), you will be asked to choose a key-length. The general rule is, the longer the better. Without getting into the details of encryption math, the security of your key increases exponentially with key length. The only potential down side to a longer key is the additional time it takes to generate and to perform its encryption operations. Nevertheless, with a Mac of recent vintage you should not notice more than a minimal lag when performing any GPG operations like encryption and decryption, even using the longest available key.

Following the selection of a key length, you will be presented with the option of specifying an expiration date. While this may be useful in some environments, it is rather limiting and therefore not recommended unless you have a specific need.

Next you will be asked to provide your identification data such as Name, Email Address and an optional comment. It is good form to use your real name as you are taking the responsibility of a self-certifying authority. So be honest.

You will then be prompted for the passphrase to protect your private key. Check over your configuration options, and

correct them via the Go Back button, or simply press Continue to generate your keypair. A lengthy keypair may take 30 minutes or more to generate, so a coffee break may be in order.

When your keypair generation has completed, you will see you new public key in the GPG Keychain Access Application (Figure 3). You can see the sub-key we spoke of earlier, by clicking on the disclosure triangle.

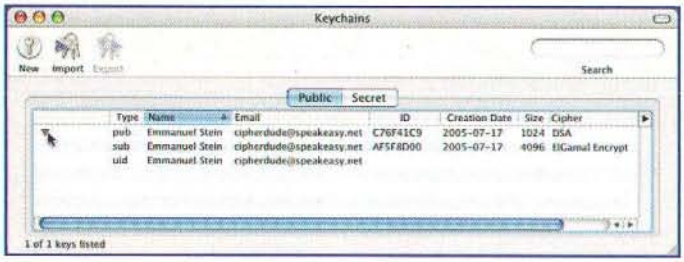


Figure 3. View of Sub-Keys Using the Disclosure Triangle

To install the GPG Preference Pane, select the Preferences option in the Application Menu. Click yes in the dialog box that pops up (Figure 4).

Figure 4. "GPG Preference Pane Not Installed" Dialog

The GPG Preferences installer will then walk you through the installation of the GPG System preference pane (Figure 5).

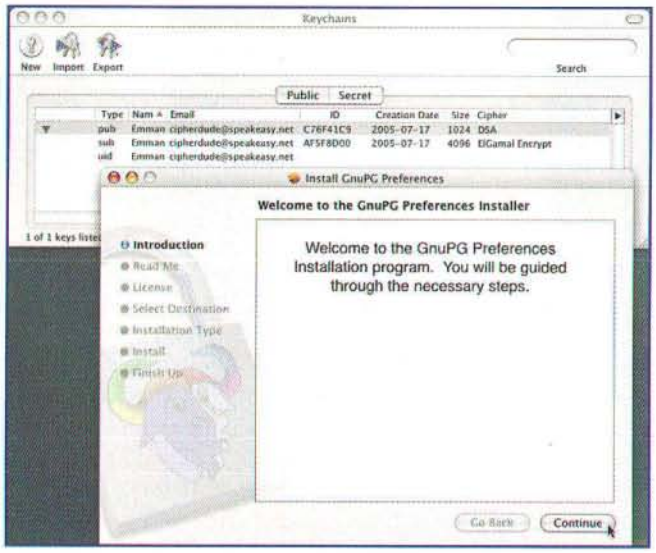


Figure 5. GnuPG Preferences Installer Invoked via the GPG Keychain Access Application

Once the GPG preference pane has successfully installed, fire up the System Preferences application and click on the GPG Preference pane to explore the various configuration options. When you first open the GPG preference pane, you will be asked to choose UTF-8 for string encoding (Figure 6). Select the default Please Do option to ensure proper GPG operations.

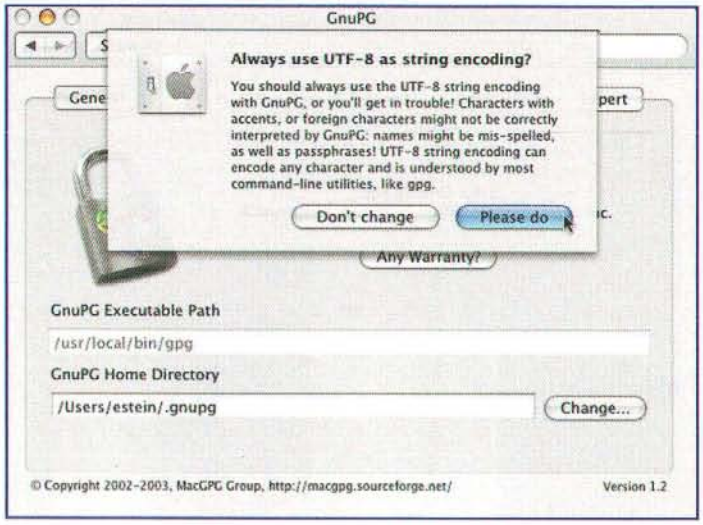


Figure 6. GnuPG Preferences Pane and UTF String Encoding Dialog

You may use this preference pane to customize GPG. Don't let all these available options overwhelm you, though, and keep in mind that the defaults are fine for most folks.

Managing your GPG Keychain

GPG relies on a centralized key management system of keyservers. Upload your public key to a GPG keyserver (private key not accepted!) to enable others to send you encrypted messages.

In the GPG Keychain Access application, click on your public key to highlight it and then select Key > Send to keyserver (Figure 7).

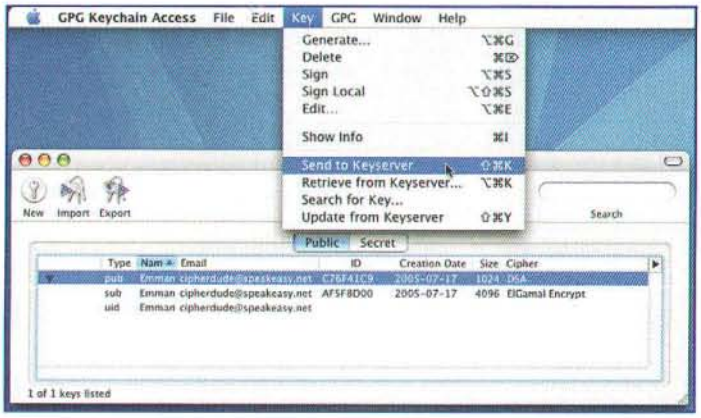


Figure 7. Send To Key Server Menu Option

Upon selection of this option, a terminal window will pop up and automatically run the terminal commands for sending the key to a keyserver for you (Figure 8).

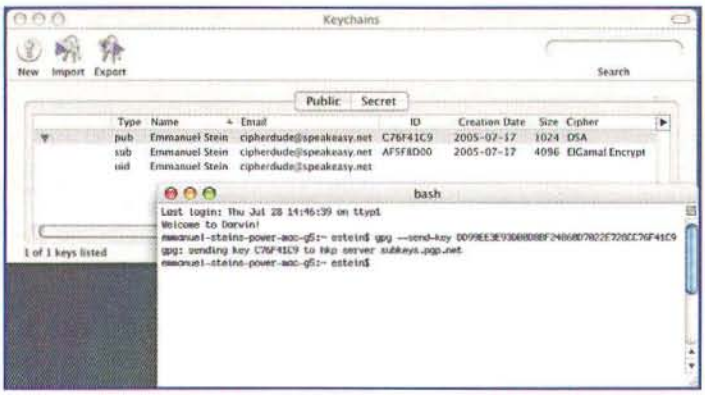


Figure 8. Invoked Terminal Session: Sending Public Key to a Keyserver

Now that you have made you public key available to others, you will need to download your correspondent's public keys using the GPG Keychain Access application. You can also download public keys via the mail interface, which will be covered in the next section.

Click Key > Search for Key (Figure 9).

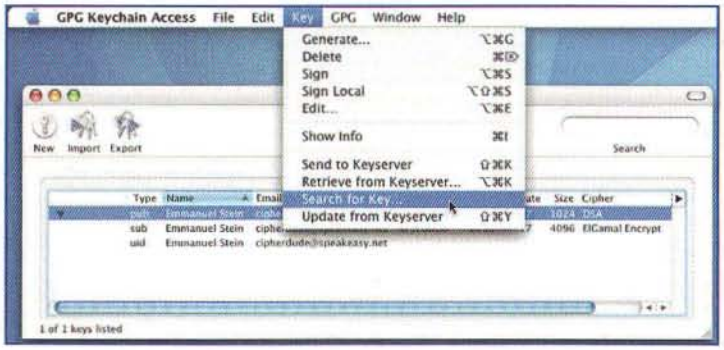


Figure 9. Search For Public Key Menu Option

You will be prompted to enter a search parameter (Figure 10), which is typically your friend's email address.

Figure 10. Search For Public Key Dialog

Once you have entered the email address and clicked on the OK button, a new terminal will appear and automatically execute the required GPG command. If your search turns up a public key, you will be prompted to download it to your GPG keychain. This step will require you to select the public key(s) for download, from within the terminal. In the illustrated example (Figure 11), we see Ed Marczak's public key associated with the email address we used in the search field. Once you make your selection, you will see output in the terminal confirming the download (Figure 11).

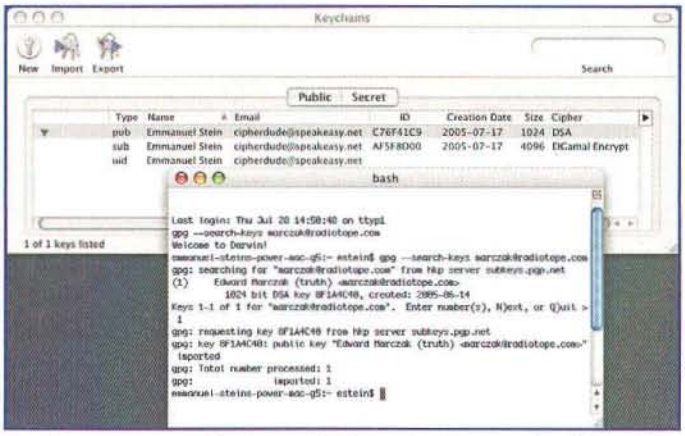


Figure 11. Invoked terminal Session: Downloading Ed's Public Key

To view the newly imported public key in the GPG Keychain Access application, go to Window > Refresh.

Once you have uploaded you public key to the GPG keyservers and downloaded your correspondent's public key(s), you are ready to install GPGmail plug-in for Apple Mail.

Apple Mail With a Side of GPG(Mail)

Double-Click on the GPGMail .dmg file to reveal the mail bundle and install script. If Apple Mail is open, be sure to quit the application before continuing. Proceed by double-clicking on the Install GPGMail AppleScript and click Run in the dialog that pops up. Once the installer finishes, a dialog box will appear. Click on the Launch Mail to start the Apple Mail.

To configure your GPG mail preferences simply go to Mail > preferences > PGP (PGP and GPG are often used interchangeably by the application's developer) shown in Figure 12.

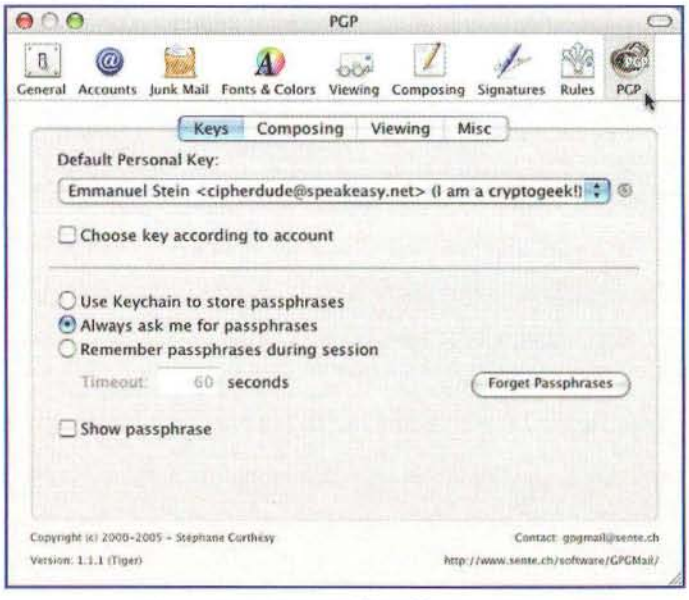


Figure 12. GPGMail Preference Pane

The default configuration should get you started, although, as you use the GPG mail bundle you may want to change several parameters to suit your use of encryption.

Changes to the Mail Interface

Once you install GPGMail for Apple Mail, you will notice three new menu items. These include GPG Keys under the View menu, which control the display of GPG attributes in Mail. (Figure 13)

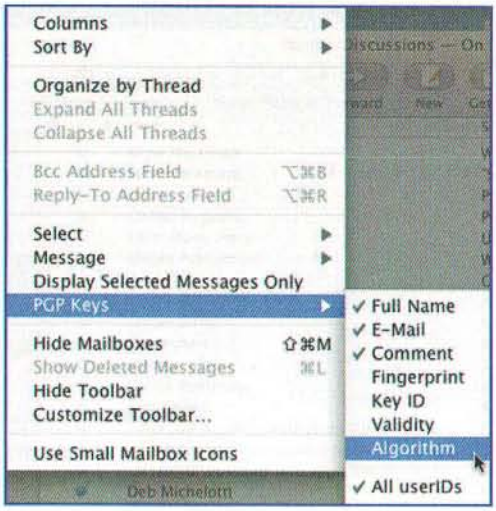


Figure 13. PGP Keys submenu in the View Menu

The second menu addition, PGP is located in the Message menu. This submenu can be used to access common GPG operations as detailed in Figure 14.

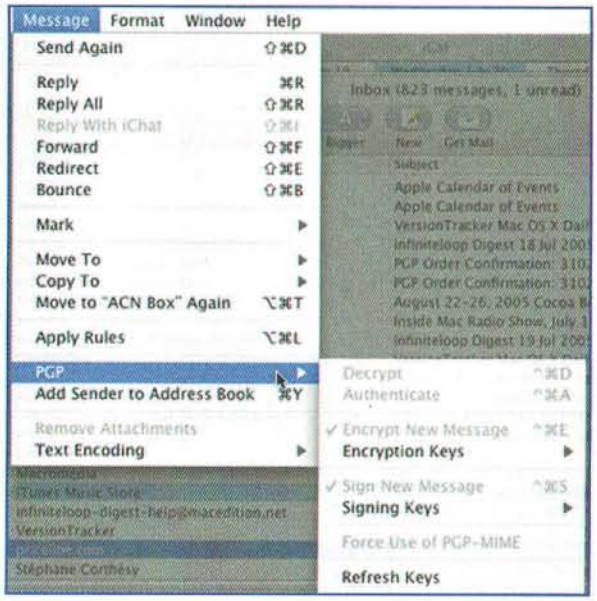


Figure 14. PGP Submenu in the Message Menu

A final added menu items is the PGP Key Search function in the Window menu, which lets you search and download public keys. This tool also pops up when you attempt to encrypt a message and send it to a recipient for whom you have not yet downloaded a public key (Figure 15).

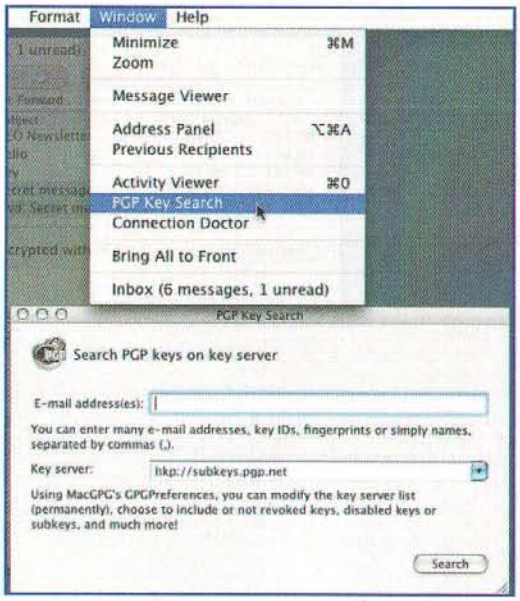


Figure 15. PGP Key Search Menu Option and Accompanying Dialog

The results will be returned in a PGP Key Search dialog box, from which you will be able to download the required key (Figure 16).

Figure 16. Downloading a Public Key in the PGP Key Search Dialog

The public keys that you download using PGP Key Search will be available globally to all GPG key management interfaces, such as the GPG Keychain Access application.

One of the most obvious changes to the mail interface is the addition of checkboxes (Figure 17) for signing and encrypting that appear in each new message window.

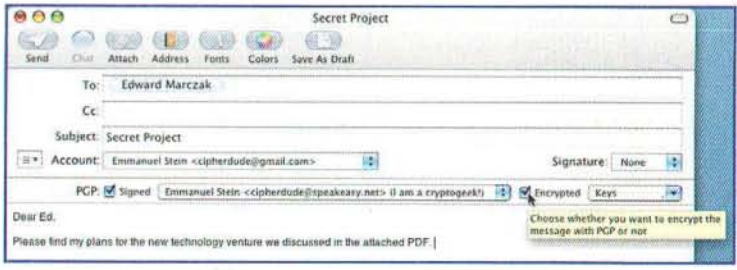


Figure 17. Check Boxes for Signing and Encrypting in an Apple Mail Message Window

When sending an encrypted mail, by default, the message will be encrypted with the public key of the recipient. However, you can explore the encryption options in the **Keys** drop down menu next to the **Encrypted** checkbox to generate, for instance, a **message password**, which uses symmetric key encryption to scramble the message and for which the recipient must know the password.

Although, I encourage you to explore all the available options, you can effectively use GPG without ever having to touch the drop-down menu or GPGMail preference pane. Just use the checkboxes, to sign and/or encrypt your communications as needed. By default, GPGMail automatically turns on the signing and also turns on encryption when it detects a recipient whose public key is in your keychain.

When you send an encrypted mail, you will be prompted for your passphrase, created during the generation of your keypair. Once it is entered, (Figure 18) your message will be encrypted and then sent.

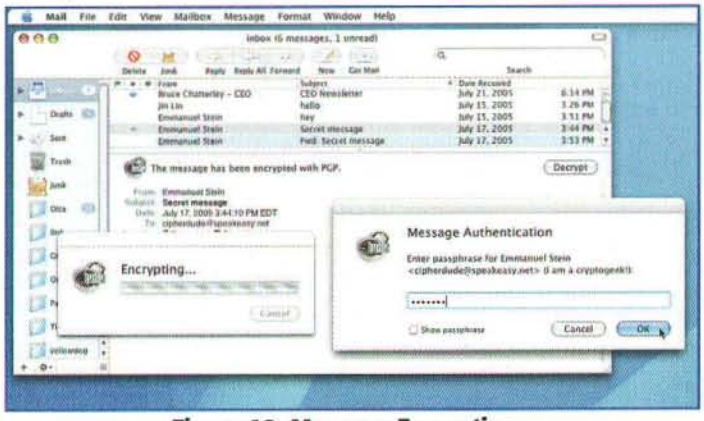


Figure 18. Message Encryption

When you receive an encrypted message, press the **decrypt** button in the message window (Figure 19). You will be prompted to enter your passphrase in order to decipher the message contents.

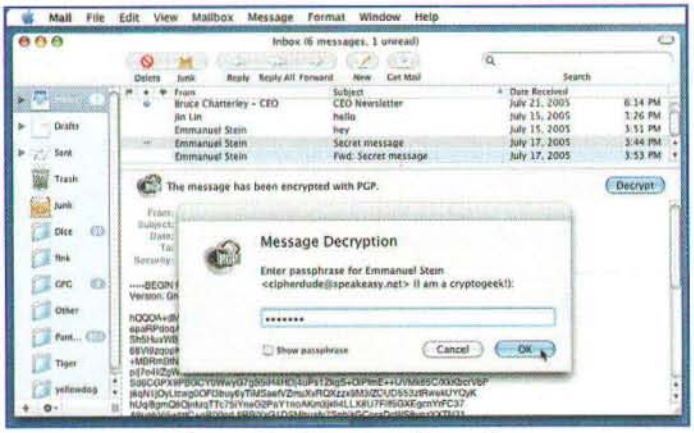


Figure 19. Message Decryption

Notice that in Figure 20, the **disclosure triangle** has been expanded to display the time and date when the message was either signed or both signed and encrypted. You may use the signing information to verify the authenticity of the message.

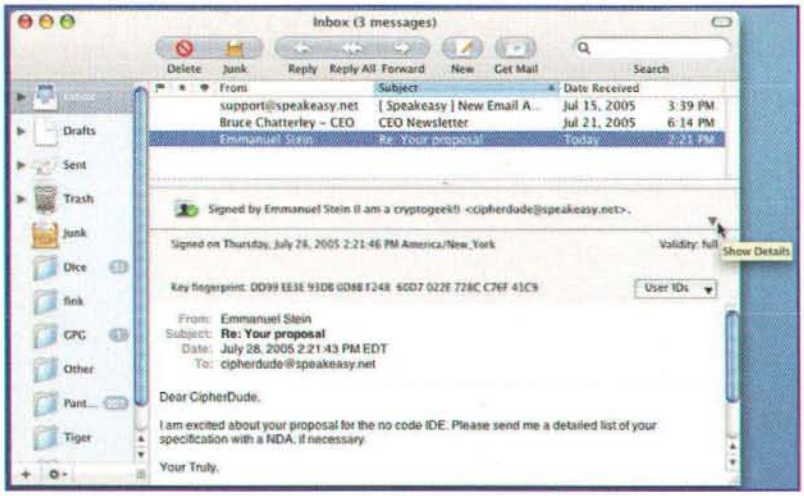


Figure 20. Decrypted Message Contents

GPGing Other Mail Clients

Although we have focused on Apple Mail 2.0 for Tiger, there are several good mail client extensions available for popular Mac mail clients. Please find the URLs listed below:

- Mozilla/Netscape Communicator/Thunderbird uses enigmail, which may be obtained at http://enigmail.mozdev.org/index.html.
- For MailSmith Users, Alex, has some great tools for encrypting and decrypting in this program; see http://alex.primafila.net/Mailsmith-GPG
- Eudora fans will find a host of useful AppleScripts for encryption and decryption at http:// mywebpages.comcast.net/chang/EudoraGPG/
- Finally, for those using Entourage X or 2004, a plug-in is available at http://entouragegpg.sourceforge.net/

Final Thoughts

The right to privacy has come a long way since 1991, when PGP was first released. With the nation in the midst of the first Gulf War, government attempts to curtail privacy and free speech threatened to suppress emerging mass encryption technologies like PGP. In fact, the Senate passed the omnibus anti crime bill (S.266) in 1991, which effectively banned the use of high-grade public key cryptography altogether!

Only through the efforts of the brave few who leaked PGP onto the Internet in 1991, partly in protest of the Senate bill banning the use of strong cryptography, did government attempts to stifle innovation and invade personal privacy become moot. So, next time you send an encrypted message (or use Apple's FileVault for that matter), tip your hat *"to the crazy ones, the misfits, the rebels, the trouble makers, the round pegs in the square holes, the ones who see things different. They are not fond of rules and have no respect for the status quo, because the people who are crazy enough to think they can change the world are the ones who actually do."* -Apple Corp., 1999

MU

About The Author

Emmanuel Stein has been an avid Mac user since 1984. He honed his cross-platform skills while working at France Telecom, Time Magazine and Reed-Elsevier. He recently started his own Mac-centric consulting company, MacVerse, which offers secure network deployments, system administration, and development services for the enterprise. A diehard Linux fan from the early days, Emmanuel enjoys hacking open source software on the Macintosh and giving lectures on the use of open source technologies on OS X. You can reach him at macverse@mac.com.

Extreme Macs

From legendary Apple guru, Scott Knaster, find out all the coolest hacks, tweaks, and mods that you can make to your Mac®, iPod® or even to Mac OS® X Tiger™

ExtremeTech™

Available wherever books are sold.

WILEY
Now you know.

DNS And E-mail Part 2: Troubleshooting

How And Where To Look For The Source Of Trouble.

You *know* that something isn't working right. Perhaps you're not receiving mail, or worse, the users have noticed before you have, and now your phone is ringing. But where do you start looking? What tools can you use to root out the problem? If you're wondering about the answers, read on.

Hello Again

Last month, we got into DNS and e-mail, and how they relate. This month is part 2, and depends on part 1. We got into some basic troubleshooting, but there's more to the equation. If you're setting things up for the first time, you tend to run into DNS related issues. If things have been up and running for a bit, and then suddenly stop, there's another range of issues to look at. But for me, it all starts in one place.

Sunday Papers

Always look at the logs. I've said it before: logs are the heart of the system. If you're a system administrator, you should always be monitoring the logs. Always. Partially by eye, partially by script that will alert you to problems. In the case of mail, the logs are going to tell you one of two things: the problem is directly on your system, the mail server, or, the problem lies elsewhere, outside of the mail server. In the former case, there's either database corruption or a simple misconfiguration. In the latter, while it can be several things, I'm betting on DNS as the culprit.

On The Outside

Let's start with this case: you're getting calls that inbound mail isn't showing up. Or, perhaps you just set up a server from scratch and noticed this for yourself. What do you do? Check the logs. Specifically, /var/log/mail.log. In one case, you'll send e-mail from an outside test account (GMail, I'm looking at ***you***), and the logs will show...nothing. Nada. No movement. Well what's wrong?

This can be one of three things: 1. Postfix just isn't running (this should be unlikely, as watchdog/launchd take great pains to make sure postfix is always going. But hey, it can happen). 2. Your firewall/router isn't allowing the SMTP protocol to reach your mail server, or 3. Outward facing DNS isn't correct.

What Can I Do?!?

In the case that Postfix isn't running, start it! It may be that simple. If you've been toying with the config files by hand (or you have some disk corruption), there may be something preventing it. How will you know? ***Check the logs***! When you start Postfix, by Server Admin or CLI, watch /var/log/mail.log. If there's a problem, it'll tell you.

In the case where your firewall isn't configured correctly, I unfortunately can't help you directly, but merely point it out as a source of problems. There are many, many, many varieties of firewall out there, along with many ways to have a network configured. I can't speak to them all. In the general sense, though, port 25 for SMTP must be able to traverse your firewall, reach your mail server, and your mail server should be able to

reply via the same path – most services aren't cool with asymmetric routing.

Then there's the case where the world just doesn't know how to send you mail.

Contact

As I touched on last month, DNS plays an important role in the delivery of e-mail. Specifically, when a mail server needs to deliver a piece of e-mail somewhere other than its local self, it queries DNS for the MX record of the recipient's domain.

If someone on the outside world is trying to send you mail, the DNS server they're using had better be able to get their mail server an MX record for your domain. If it can't, their mail will sit in their local queue for a bit. The tool for this job is dig – the domain information groper.

By default, dig will lookup DNS 'A' records, similar to the now deprecated nslookup:

```
$ dig www.example.com

; <<>> DiG 9.2.2 <<>> www.example.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8947
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2,
ADDITIONAL: 0

;; QUESTION SECTION:
;www.example.com.          IN    A

;; ANSWER SECTION:
www.example.com.  172800  IN    A        192.0.34.166

;; AUTHORITY SECTION:
example.com.      21600   IN    NS       a.iana-
servers.net.
example.com.      21600   IN    NS       b.iana-
servers.net.

;; Query time: 4172 msec
;; SERVER: 192.168.100.12#53(192.168.100.12)
;; WHEN: Tue Jul 19 23:56:45 2005
;; MSG SIZE  rcvd: 97
```

Gives us quite a bit more information than nslookup, too. The quick way to look at this is:

Question section: what was asked of us?
Answer section: the answer to the query, if possible.
Authority section: which servers are authoritive for the domain in question.

We, however, need to check the MX record for our site. Let's look up a domain that has a nice, clean example. Here's an example from my favorite example, GMail:

```
$ dig MX gmail.com

; <<>> DiG 9.2.2 <<>> MX gmail.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12756
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 4,
ADDITIONAL: 4

;; QUESTION SECTION:
;gmail.com.                IN    MX

;; ANSWER SECTION:
gmail.com.      3600    IN    MX  10 gsmtp171.google.com.
gmail.com.      3600    IN    MX  10 gsmtp185.google.com.
gmail.com.      3600    IN    MX  10 gsmtp171-
2.google.com.
gmail.com.      3600    IN    MX  10 gsmtp185-
2.google.com.
gmail.com.      3600    IN    MX  5 gmail-smtp-
in.l.google.com.

;; AUTHORITY SECTION:
gmail.com.      86400   IN    NS       ns3.google.com.
gmail.com.      86400   IN    NS       ns4.google.com.
gmail.com.      86400   IN    NS       ns1.google.com.
gmail.com.      86400   IN    NS       ns2.google.com.

;; ADDITIONAL SECTION:
ns1.google.com. 276050  IN    A        216.239.32.10
ns2.google.com. 276050  IN    A        216.239.34.10
ns3.google.com. 276050  IN    A        216.239.36.10
ns4.google.com. 276050  IN    A        216.239.38.10

;; Query time: 46 msec
;; SERVER: 192.168.100.12#53(192.168.100.12)
;; WHEN: Tue Jul 19 23:19:04 2005
;; MSG SIZE  rcvd: 306
```

First thing to point out: with dig, you can simply give it the record type that you're after. Back to e-mail: notice that in a proper setup, the answer section contains all mail servers, along with their priorities. Remember: the lower the number, the ***higher*** the priority. Also notice that in the answer section, each record ends with the root node – the dot ("."). gmail-smtp-in.1.google.com is the highest-priority mail server. All mail deliveries should be attempted there first. This is followed by 4 backup servers that have equal priority. If the main server is down or unreachable, outside mail servers will choose one of these to deliver to. The algorithm for this is actually a little more complex than you'd think at first look – it doesn't simply choose one at random. This algorithm is part of a deeper discussion, though. Once the main server is back up, they will hand off the mail that they've queued up.

Naturally, you ***should*** have a backup mail server, and it ***should*** reside at a separate location on a separate network. If you can't configure this yourself, your ISP will very often act as a backup MX for you. I've also helped with some creative arrangements where companies will backup each other. Just be aware that mail that gets queued on a backup sits there unencrypted, so you need to trust your backup partner. This is, actually, a good reason to start encrypting e-mail yourself. Check out the GPG article in this issue by Emmanuel Stein for instructions on just that. Additionally, I have an introduction to GPG, digital certificates and why I sign on my site at http://www.radiotope.com/writing/?p=13.

dig is happy to query other DNS servers. In the case of mail, you need to do this. Make sure you're querying a server outside of your LAN so you're getting the same perspective as the world. Run it like this:

```
dig @199.184.165.1 MX yourdomain.com
```

(199.184.165.1 is an actual DNS server – be kind: don't hammer servers that are open to the public. Find and use the DNS server provided by your ISP.)

If DNS *is* the culprit, what's wrong? Did the foray into dig shine a light on the issue? Were there any surprise or missing entries? Malformed entries? Mail server priorities set correctly? As you can see, there are a lot of 'i's to dot and 't's to cross. Computers can be sensitive to that kind of thing.

If your DNS is correct, try the telnet trick from last month's column. Either mail is getting through, or it isn't. If mail can't penetrate a firewall, correct DNS won't make a difference.

Inside

If your setup passes the DNS test, what's next? To the logs! You should still be looking at /var/log/mail.log – preferably in a terminal using "tail -f" so you're watching it update in quasi-real time.

This is a quick refresher regarding the e-mail subsystem. If you need more depth, please refer to part 1 of this article.

The first thing to remember is this: 'e-mail', although typically viewed as one single entity, is traditionally many separate pieces. In our case, Apple uses Postfix for smtp – the server receiving e-mail from the outside world, and from your mail client, and they use Cyrus for POP and IMAP – your mail client checking mail. All of these components have convenient acronyms to go along with them (Woo! More acronyms!):

- MTA: Mail Transfer Agent – your SMTP server, as it's responsible for transferring mail.
- MUA: Mail User Agent – this is your e-mail application such as Eudora, Mail.app or Pine. It's the user interface into the mail store.
- MDA: Mail Delivery Agent – This is a program that is responsible for getting the mail from the MTA and dropping it in the mail store.

Traditionally, your MTA (such as sendmail) would receive mail (internally or externally), and know how to get it into the system mail spool all by itself. The system mail spool contained mailboxes for each user on the system, and it would do so using mbox format. mbox is the traditional Unix mailbox format. It stores all mail for a single mail folder, like your inbox, in a single, large text file. As you can imagine, this doesn't scale too well. How many people reading this article have, or know someone who has, an inbox that's over 500MB? How about 1GB? Yeah, it happens. To keep all of that in a single, un-indexed text file can cause you some performance issues. On the other end of this, a MUA, like 'mail' or pine, would know how to reach into that same mail store and display mail to the end user. Cyrus changes all of that, so this discussion will focus on what we *are* dealing with.

Subdivisions

If you've never touched Apple's stock mail config, you should very rarely see problems. Now that the log rolling bug has been fixed, you almost have to *try* to destroy Cyrus. See part 1 on ways to detect and troubleshoot Cyrus issues. Postfix is typically just a rock. But we're techs, right? We can't leave well enough alone! We can improve it! Or, more likely, your employer or your client ask you about some functionality that doesn't already exist in Apple's config. Thankfully, 10.4 Server brings us integrated Amavis with ClamAV and SpamAssassin. However, the now oft-most requested item – vacation replies – is something you have to deal with. (This should be handled by sieve, but it was broken under 10.3, and currently remains so under 10.4.2).

While Postfix has several tables and files, two really cover the vast majority of configuration: main.cf and master.cf. These are stored along with the rest of Postfix's configuration files in /etc/postfix. If you are ever tempted to make changes to these files, back them up first! It'll take you two seconds to tar them up (tar czvf ~/postfix-`date '+%Y%m%d%H%m'`.tar.gz /etc/postfix), or just simply copy them. You'll be much happier that you did when things stop working, and you need to go back to things the way they were.

main.cf is the global configuration file for Postfix. When you edit information in Server Admin (or Postfix Enabler), you're altering this file. main.cf is primarily responsible for defining the service's role:

What domain to receive mail for.
What destinations to relay mail to.
How to deliver mail (direct or relay).
The domain to use in outbound mail.
Clients to allow relay from.

Apple includes a default main.cf file in /etc/postfix. Server Admin just tacks the parameters that it changes onto the end of the file. So you'll find two values for certain parameters. The second will override the first, so Apple's method is OK. In some ways, it's nice that all of the values it changes are grouped.

The format of main.cf is simple: parameter = value. Like a unix shell variable, you can use the value of the parameter later on in another parameter like this: parameter = $other_paramter. Interestingly, since Postfix does not compute the value until run time, you can use a variable before you actually assign anything to it. The lesson here is that one reason Postfix may not start is that main.cf is mangled. The one parameter that ***must*** be right here is "mydestination" – this should contain all domain names that Postfix will accept as local, and hand off to 'deliver'.

Any time you make changes to the configuration files, you must issue a postfix reload for Postfix to pick up the changes. I've seen people make changes, and then wonder why the changes aren't working. No postfix reload, no changes.

More likely, you find some neat addition to Postfix, and the instructions ask you to modify master.cf. master.cf controls Postfix's master process, which in turn controls all Postfix child processes.

Each line in master.cf defines a service in Postfix. The order doesn't matter, however: a) if you define a service multiple times, only the last one is honored, and b) it's smart to keep logical groups together, for your own sanity. Empty lines and comment lines (begin with '#') are ignored. A logical line begins by having non-whitespace text in column 1. A line is continued by having white-space precede the text. Each line has 8 columns. The man page can do a better job of explaining the basics than I can. Unfortunately, Apple doesn't include this man page, so go check out http://www.postfix.org/master.5.html.

The part that tends to throw people off is taking generic installation instructions and modifying them for OS X. Sometimes an installer will try to put a binary or script in a location that is not in your path. Postfix filters should ***never*** be run as root, so you'll need an additional system user to run filters. However, most generic instructions tell you to add a user using adduser – present on all Unicies but OS X (not by default, anyway).

There are three ways to get a filter to run: globally, by placing a content_filter= statement in main.cf, by specifying a content filter for a given path (master.cf), or, by running a filter on the fly, based on some criteria (altering lookup tables or checks).

The first method is the most simple, and is used by Apple to hook Amavis into Postfix. Note the last line of main.cf on a Tiger Server:

```
content_filter = smtp-amavis:[127.0.0.1]:10024
```

This makes Postfix run all queued mail through a filter called smtp-amavis. How does Postfix know what "smtp-amavis" is? It must be defined as a service in master.cf. You'll see that master.cf has this definition (under Tiger):

```
smtp-amavis unix -    -    y    -    2    smtp
```

This content filter line in main.cf will take ***all mail*** and run it through the filter "smtp-amavis", and that's really what we want for an anti-virus filter. However, if, for some reason, we wanted this filter to run only for incoming mail, we have to remove the content_filter statement from main.cf (that's global, remember – in ***and*** out) and redefine the smtp service to have two unique instances: one for inbound and one for outbound. This needs to be done by port, or by IP address. This tends to confuse people that are running with one physical interface.

The easiest way to handle this is to have two IP addresses. You either need two physical interfaces, or you can multi-home a single interface. Then, you can add a line in master.cf that applies only to the outbound interface:

```
<outbound ip>:smtp    inet    n    -    y    -    -    smtpd
    -o content_filter=smtp-amavis:
```

The "-o" flag overrides a main.cf parameter. In this case, we'll only filter if mail is traveling over the outbound ip. Also note: there is never any white-space surrounding a "=" in master.cf. The trailing ":" after the filter name is important! When left blank afterwards, it means 'for all domains'.

Another way to handle different routes for different filters is to use different ports. Naturally, this is dependant on the filter itself, and the protocol involved. For example: while server to server smtp would be tough to change from the default of port 25, you can certainly tailor your submission port to suit your needs.

The third way is a bit more dynamic. Postfix will let you filter on the fly by using the access table or header_check and body_check rules.

A great feature of Postfix is that it performs filtering ***after*** mail is in the queue, but before it gets delivered. This way, if there's a problem with the filter, mail doesn't bounce, but simply gets queued up. You'll see this in the logs as "status=deferred". Check your mail queue by typing "mailq" at the command line. You'll see something like this:

```
# mailq
?-Queue ID- -Size- --Arrival Time-- -Sender/Recipient--
- ?8EF0949EA 710 Tue Jan 13 20:43:31 ?
710 Tue Jan 13 20:43:31 ?
(deferred ?transport) ?
- 1 Kbytes in 1 Request.
```

In the case where you see "deferred", you need to take care of the filter, or remove it by commenting it out of both main.cf and master.cf. Remember from last month, a 'deferred' condition can also apply to Cyrus' deliver agent not being able to, er....deliver!

If you can fix your filter (a socket based filter may simply need to be restarted, a pipe based filter may need some re-coding), great. You should simply have to postqueue -f to flush the queue and have Postfix make re-delivery attempts.

If you've removed the filter (temporarily, right?), you need to re-queue the mail, otherwise it will keep trying to deliver itself through the old (now non-existent) channel. Do this, as root (or use sudo), with the postsuper command: postsuper -r ALL, followed by a postfix reload. Watch the logs and your mail should start flowing freely. Follow that up by another mailq.

Learning To Fly

This isn't the end – we're just learning to fly. Hopefully these troubleshooting guides help you find some of the trouble in an e-mail system should you run into it. As I say each month: watch the logs! This is the heartbeat of your system, and it'll let you know immediately when there is a problem.

Naturally, there can be other problems that crop up in the OS X mail system: mailman issues, other custom work or installations that may grab a port on you, etc. The best way to learn how something works is to watch it fail – and then reinforce that by diving in and fixing it!

Next month, we're going to take a break from e-mail and DNS, and get back to something a little more straight-forward: the Terminal, and one of my favorite utilities, "screen".

P.S. :
Johnny Cash – Hello Again
Joe Jackson – Sunday papers
Oingo Boingo – On the Outside
Ice Cube (not .38 Special) - What can I do?
Stereolab – Contact
Buzzcocks - Inside
Rush – Subdivisions (to all you ACNs that landed at YYZ for camp!)
Pink Floyd – Leaning To Fly

MT

About The Author

Ed Marczak, owns and operates Radiotope, a technology consulting company that implements mail servers and mail automation. When not typing furiously, he spends time with his wife and two daughters. Get your mail on at http://www.radiotope.com

DEV DEPOT

DevDepot has it all!

Get More out of your Mac!

DevDepot sells the tools, toys and technology to put more muscle into your Mac. Visit our online store today for special offers and great new products. **www.devdepot.com**

Television is about to get a whole lot more interesting.

NOW ONLY $324.99

EyeTV 200

The future of Television

Watch TV on your Mac!

Watch TV on your Mac

EyeTV features a 124-channel cable-ready analog tuner and DVD quality MPEG-2 video encoder.

Record TV on your Mac and Archive to Disk

Collect and organize your favorite shows to **watch whenever you want**, then update video content to DVD.* (Toast 6 Titanium required).

Don't miss a thing

Use EyeTV's Electronic Program Guide to find exactly what you are looking for, and program EyeTV to record it. **Program EyeTV from anywhere via the Internet.**

Features and Benefits

- Record TV otn Your Mac
- Edit Out Unwanted Content
- Archive Recorded TV To DVD*
- MPEG-2 Video Encoding
- Digitize Analog Video
- The Speed And Power Of **Firewire**

go digital

Convert your analog video tapes to digital in realtime, then burn them to DVD! *

NEW!

New Export Functionality

EyeTV now lets you export to iMovie®, iDVD® and DVD Studio Pro®, making it easier to create professional quality recordings.

from elgato

Palm On X

Dissecting The PDB File

By Jose R. C. Cruz

Introduction

Welcome to the debut of **Palm On X**. This is the first of a series of articles dedicated to using MacOS X as a development platform for PalmOS ®. I hope that the information I present in these articles would help inspire others (including myself) to independently provide PalmOS development tools and utilities that take advantage of such MacOS X technologies such as Cocoa, Core Foundation and XCode.

These articles are not meant to teach PalmOS application development only. Those who want to learn PalmOS programming may be interested in checking out ***Palm Programming: The Developer's Guide*** written by Rhodes and McKeehan and published by O'Reilly.

What is a PDB file

PDB stands for **P**alm **D**ata**B**ase. It is a binary file format used by the PalmOS as the means to store data on a PDA handheld. It is designed to make efficient use of limited storage space while supporting a large variety of data types. It also allows the storage of metadata that is specific to a PalmOS application.

Structure of a PDB file

Figure 1 shows the general layout of the PDB file. The file itself is subdivided into 5 distinct data blocks: Header, Record List and Record Entry, Application Information, Sort Information, and Record Data.

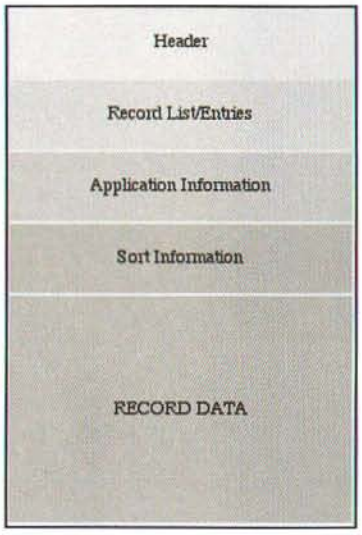


Figure 1: General layout of the PDB file.

The Header Block

The Header block contains data describing the entire PDB file. It stores the name, version, database attributes, creation and modification dates, type and creator signatures, and offsets to any application-specific data.

The basic layout of the Header block is shown in Figure 2. Listing 1 shows the data structure that defines the block. The elements that comprise the DatabaseHdrType datatype are as follows:

- name. The name of the file as a 32 character C-string .
- attributes. The attributes associated with the file.
- version. The version number of the file.
- creationDate. Date when the file was last created.

- modificationDate. Date when the file was last modified.
- lastBackupDate. Date when the file was last backed up.
- modificationNumber. The modification ID number assigned to the file by the PalmOS.
- appInfoID. The offset of the optional AppInfo data block from the beginning of the file.
- sortInfoID. The offset of the optional SortInfo data block from the beginning of the file.
- type. The four-character signature assigned to the file.
- creator. The four-character signature of the PalmOS application that created the file.
- uniqueIDSeed. The unique ID number assigned to the file by the PalmOS.

The date elements are expressed in the number of seconds since 1970 January 1. This is true for PDB files that were created on PalmOS 4.x or later. Older files may use a different reference year (1900 or 1904), especially if the files were synchronized on a non-Windows platform. In either case, the PDB date values will have to be converted since MacOS X uses 2001 as its reference year.

The attributes element consists of 16 bit-flags. Each flag indicating how the PDB file is to be handled by the PalmOS system. Shown in Figure 3 is a breakdown of each bit flag and its significance.

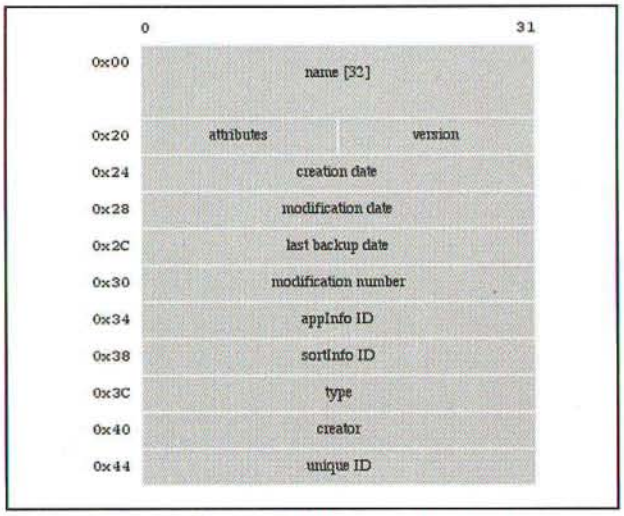


Figure 2: Layout of the Header block.

Listing 1. Data structure of DatabaseHdrType.

```
struct
{
    unsigned char   name[32];
    unsigned short  attributes;
    unsigned short  version;
    unsigned long   creationDate;
    unsigned long   modificationDate;
    unsigned long   lastBackupDate;
    unsigned long   modificationNumber;
    unsigned long   appInfoID;
    unsigned long   sortInfoID;
    unsigned long   type;
    unsigned long   creator;
    unsigned long   uniqueIDSeed;
}   DatabaseHdrType;
```

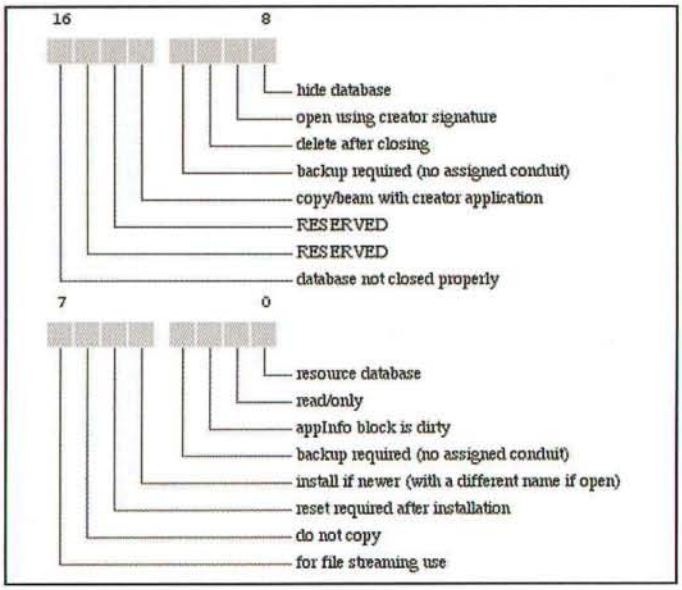


Figure 3. Attribute bit flags used in the Header block.

The Record List and Entries Blocks

Located immediately after the Header block is the Record List block. This block contains the number of record entries present in the PDB file and the location of the first record entry.

The basic layouts of the Record List and Entries blocks are shown in Figure 4. Listing 2 shows the data structure that defines the Record List block. The elements that comprises the RecordListType datatype are as follows:

- nextRecordListID. Pointer to the next RecordListType structure. This pointer is updated by the PalmOS only when the current RecordListType structure was unable to add more items due to memory constraints. The default value is often 0x00000000.
- numRecords. The number of record entries present in the block.
- firstEntry. The upper two bytes of the first record entry in the PDB file. If there are no record entries, this element is assigned the value of 0x0000 to preserve a 4-bytes alignment.

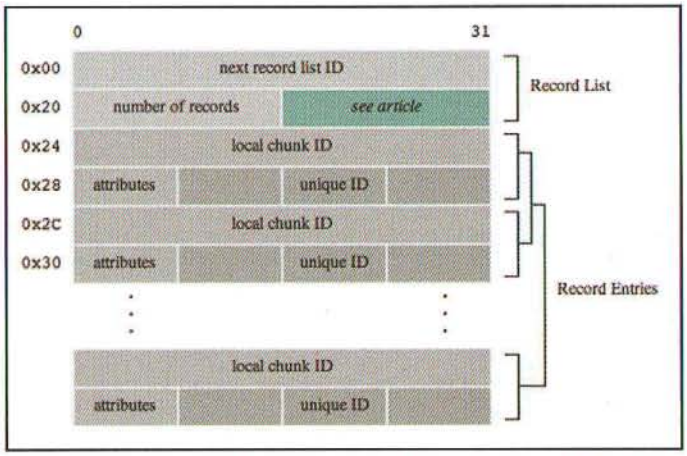


Figure 4. Layout of the Record List and Entries blocks.

Listing 2. Data structure of RecordListType.

```
struct
{
     unsigned longnextRecordListID;
     unsigned short numRecords;
     unsigned short firstEntry;
}   RecordListType;
```

The Record Entry block exists only if the numRecords of the Record List block is a non-zero value. If present, this block is located immediately after the Record List block. It is also terminated with 0x0000 after the last record entry to maintain a 4-byte alignment.

A non-zero value for numRecords does not always mean that there are valid records present. For performance reasons, some PalmOS applications assign a number of NULL record entries in their PDB files to serve as placeholders. It is often a good idea to validate the offsets for each record entry prior to reading the actual record data.

Each entry in the Record List entry block is defined by the RecordEntryType datatype. Listing 3 shows the data structure of that datatype. The elements that comprise the datatype are as follows:

- localChunkID. The offset of each raw record data from the beginning of the PDB file.
- attributes. The attributes of each record data
- uniqueID. A three-byte unique ID number assigned to each entry by the PalmOS.

The attributes element consists of 8 bit-flags. Each flag indicating how each PDB record data is to be handled by the application and by the PalmOS system. Figure 5 shows a breakdown of each bit flag and its significance.

Listing 3. Data structure of RecordEntryType.

```
struct
{
     unsigned long localChunkID;
     unsigned char attributes;
     unsigned char uniqueID[3];
}   RecordEntryType;
```

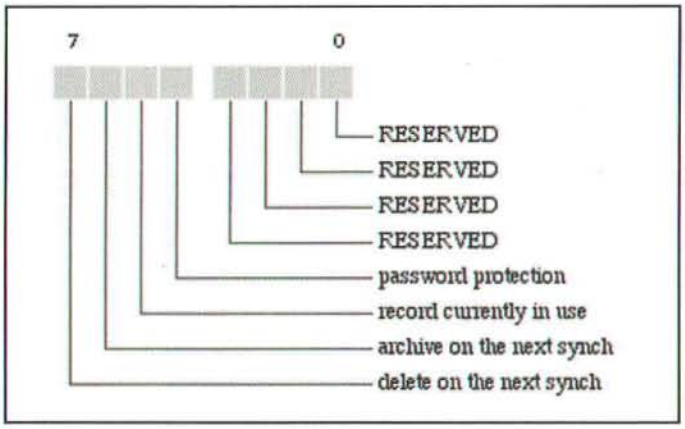


Figure 5. Attribute bit flags used in the Record Entry block.

The Application Info Block

The Application Info or AppInfo block is where a PalmOS application can store application-specific data. It is also where the PalmOS stores category information that allows users to group records into different categories. This latter feature is only available if the application itself supports the PalmOS Category APIs.

If the PDB file does not have an AppInfo block, the appInfoID element of the Header block will have a 0x00 value. However if the application does support the Category APIs, its PDB file would then have an AppInfo block with the category information located at the beginning of that block.

Figure 6 shows the basic layout of the AppInfo block with the category sub-block included. Listing 6 shows the data structure of the AppInfoType datatype that defines sub-block.

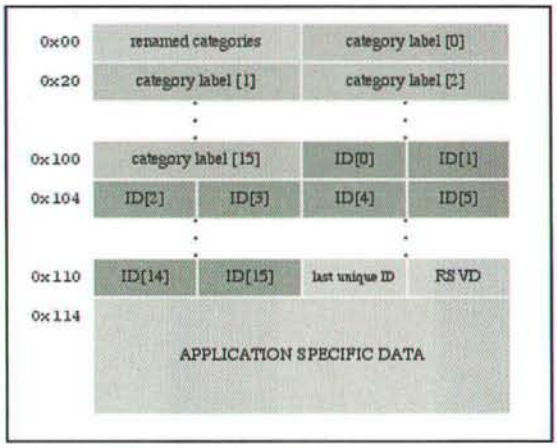


Figure 6. The category region of the AppInfo block.

Listing 6. Data structure of AppInfoType.

```
struct
{
     unsigned short renamedCategories;
     unsigned char categoryLabels[16][16];
     unsigned char categoryUniqueIDs[16];
     unsigned char lastUniqueID;
     unsigned char RSVD;
}   AppInfoType;
```

The elements that comprise the AppInfoType are as follows

- renamedCategories. The number of category labels renamed by the user.
- categoryLabels. An array of 16 category labels. Each label has a maximum of 16 characters in length
- categoryUniqueIDs. An array of 16 unique ID number assigned to each category label by the PalmOS.
- lastUniqueID. The unique ID number of the *last* category label that was used

The region between the category sub-block and the next data block will contain data specific to the PalmOS application. It is up to the developer to define the data structure of the information stored within that region. Without knowing the

exact data structure, the information contained in the application-specific portion of the AppInfo block can only be viewed as a simple hex dump.

To determine the size of the application-specific region, I use the following equation:

$$Size_{appInfo}^{app-specific} = \texttt{sortInfoID} - \texttt{appInfoID} - \text{sizeof}(\textbf{AppInfoType})$$

The preceding equation is correct only if the next data block happens to be a SortInfo block. If a SortInfo block does not exists, I use the following equation to determine the size of the application-specific region in the AppInfo block:

$$Size_{appInfo}^{app-specific} = \texttt{firstRecordEntry} - \texttt{appInfoID} - \text{sizeof}(\textbf{AppInfoType})$$

The Sort Information block

The Sort Information or SortInfo block is another area in the PDB file where a PalmOS application can store application-specific data. Like the AppInfo block, the PDB file will have a SortInfo block if the sortInfoID element in the Header block has a non-zero value. A value of 0x00 would mean the absence of a SortInfo block. However, unlike the AppInfo block, the SortInfo block is not used at all by the PalmOS (as of this writing).

The data format of the SortInfo block also differs from one application to another. Without knowing the exact data structure, the information contained in the SortInfo block can only be viewed as a simple hex dump.

To determine the size of the SortInfo block, I use the following equation:

$$Size_{sortInfo} = \texttt{firstRecordEntry} - \texttt{sortInfoID}$$

The Record Data Block

The Record Data block contains all the records stored in the PDB file. The format of each record data differs from one application to another. Without knowing the exact data structure of each record, the raw record data can only be viewed as a simple hex dump.

The location of each record in the Record Data block is stored in the localChunkID element of the Record Entry sub-block. To determine the size of each record, I calculate the difference between the localChunkID of the current record and the localChunkID of the next one as follows:

$$Size_{sortInfo} = \texttt{firstRecordEntry} - \texttt{sortInfoID}$$

However, if I want the size of the last record in the Record Data block, I calculate the difference between the size of the entire PDB file and the localChunkID of the last record as follows:

$$Size_{record} = Size_{PDB} - \texttt{localChunkID}_{current}$$

Mapping the PDB Data Into A Plist File

Exporting the PDB file data into an XML provides a number of advantages, the most notable of which are access and portability. For example, if I convert a PDB file into an equivalent XML file, I can edit the XML data using any text editor. I can also upload the XML file to any system (such as Windows, Mac, Unix, and so on) with no loss of information.

The most prevalent XML file format used on a MacOS X system is the plist file. This format uses a key-value system to store and differentiate its data. It is also relatively easy to generate. This is the format that I will use to export my PDB file data.

I defined five major keys for the plist file. Each key represents a data block in the PDB file. To ensure that each key name is unique, I use a reverse URL naming scheme similar to that used for Java classes. Note that the acronym PSRC is the ticker symbol for PalmSource, Inc.

Listing 7 is the XML structure for the com.psrc.pdb.header dictionary. This structure is a modified implementation of DatabaseHdrType (see Listing 1). Notice that I gathered all the date information under the dictionary key named date. I also gathered the appInfoID, sortInfoID and uniqueIDSeed data values into a sub-dictionary group called ID.

Listing 7. XML structure of com.psrc.pdb.header.

```
<key>com.psrc.pdb.header</key>
<dict>
  <key>name</key>
    <string></string>
  <key>attributes</key>
    <integer>0</integer>
  <key>version</key>
    <integer>0</integer>
  <key>date</key>
  <dict>
    <key>creation</key>
      <date>2005-05-18T19:29:44Z</date>
    <key>lastBackup</key>
      <date>2005-05-18T19:30:03Z</date>
    <key>modification</key>
      <date>2005-05-18T19:29:51Z</date>
  </dict>
  <key>modificationNumber</key>
    <integer>0</integer>
  <key>ID</key>
  <dict>
    <key>appInfo</key>
      <integer>0</integer>
    <key>sortInfo</key>
      <integer>0</integer>
    <key>uniqueSeed</key>
      <integer>0</integer>
  </dict>
  <key>type</key>
    <string></string>
  <key>creator</key>
    <string></string>
</dict>
```

Listing 8 is the XML structure for the com.psrc.pdb.record.list dictionary. This structure is a straightforward implementation of RecordListType (see Listing 2).

Listing 8. XML structure of com.psrc.pdb.record.list.

```
<key>com.psrc.pdb.record.list</key>
<dict>
  <key>nextRecordList</key>
    <integer>0</integer>
  <key>numRecords</key>
    <integer>0</integer>
```

```
  <key>firstEntry</key>
    <integer>0</integer>
</dict>
```

Listing 9 is the XML structure of the com.psrc.pdb.record.entry array. Each element in the array is a dictionary whose structure is an XML representation of RecordEntryType (see Listing 3). The order of each dictionary entry is unimportant. I can easily reconstruct the original order of each record entry in the PDB file by simply examining the value stored in the localChunk key.

Listing 9. XML structure of com.psrc.pdb.record.entry.

```
<key>com.psrc.pdb.record.entry</key>
  <array>
    <dict>
      <key>ID</key>
      <dict>
        <key>localChunk</key>
          <integer>0</integer>
        <key>unique</key>
          <integer>0</integer>
      </dict>
      <key>attributes</key>
        <integer>0</integer>
    </dict>
    ...
  </array>
```

Listing 10 is the XML structure of the com.psrc.pdb.info dictionary. This structure is an amalgam of both the AppInfo and SortInfo blocks. The appInfo dictionary is a straightforward implementation of AppInfoType (see Listing 6). The categories key is an array of dictionaries, each dictionary containing the values stored in the categoryLabel and categoryUniqueID elements of AppInfoType. By combining these values into a dictionary, I am able to preserve the one-to-one correspondence between category label and unique ID.

Any application-specific data found in the AppInfo block is stored using the <data></data> XML tag. I also used the same approach to store any application-specific data found in the SortInfo block.

Listing 10. XML structure of comp.psrc.pdb.info.

```
<key>com.psrc.pdb.info</key>
  <dict>
    <key>appInfo</key>
    <dict>
      <key>categoryRenamed</key>
      <integer>0</integer>
      <key>lastUniqueID</key>
      <integer>0</integer>
      <key>categories</key>
      <array>
        <dict>
          <key>id</key>
          <integer>0</integer>
          <key>label</key>
          <string></string>
        </dict>
        ...
      </array>
      <key>appSpecific</key>
      <data></data>
    </dict>
    <key>sortInfo</key>
    <data></data>
</dict>
```

Finally, Listing 11 is the XML structure of the com.psrc.pdb.record.data array. Like the sortInfo key, I use the <data></data> XML tag to store the raw record data exported from the PDB file.

Listing 11. XML structure of com.psrc.pdb.record.data.

```
<key>com.psrc.pdb.record.data</key>
  <array>
    <data></data>
    ...
    <data></data>
  </array>
```

Building A PDB Viewer

I will now show you how to design and develop a simple PDB Viewer using Cocoa. I used XCode 1.5 to develop the Cocoa application. My development system is an iBook G3/600 MHz unit running MacOS X 10.3.9.

I personally try to avoid using any features that are specific to one version of the OS when I design my applications. This allows me to maintain some level of cross-platform compatibility. It also gives me the freedom to port my source code using other development IDEs such as Metrowerks Codewarrior or Project Builder.

The User Interface Design

The UI design for the PDB Viewer is quite straightforward. I've decided to use a single window to display the information retrieved from the PDB file. I then use an NSTabView object to create four panel views. Each panel view is dedicated to each one of the PDB data blocks. Each view is also assigned a controller that will provide the necessary information to be displayed. I will talk about the controllers later in this article.

Since I am developing a data viewer, I've decided to disallow on-screen editing. On-screen editing is beyond the scope of this article. However, I do plan to revisit the concept in a future article.

The layout of the Header panel view is shown in Figure 7. This view will display the data contained in the PDB Header block. The controller assigned to this view is PDBHeader.

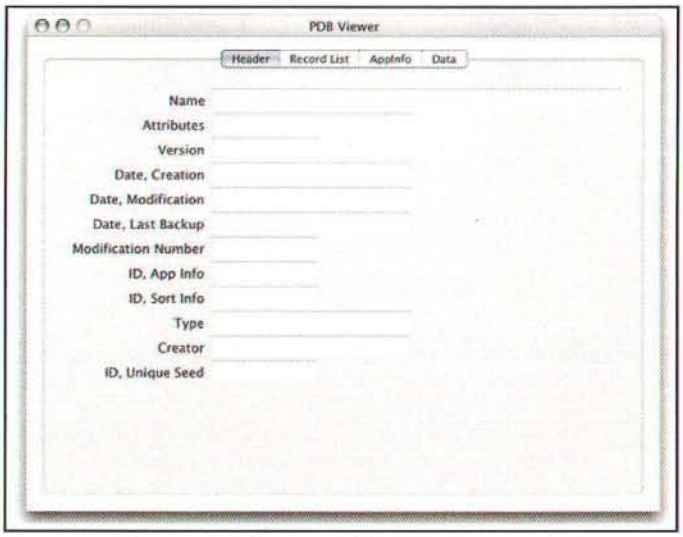
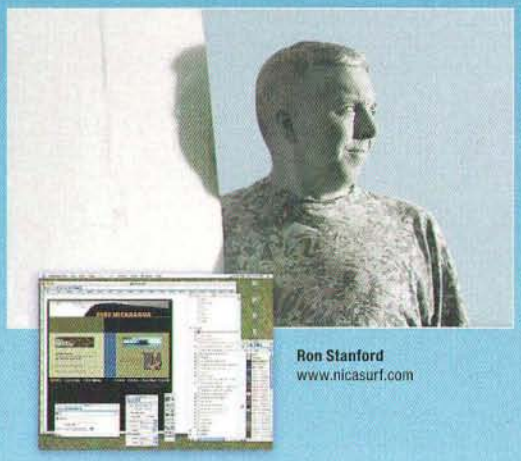
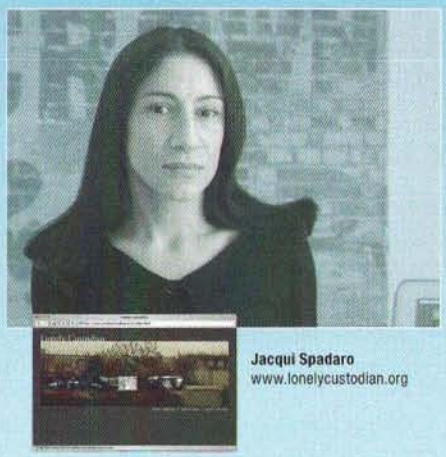


Figure 7. UI layout of the Header panel.

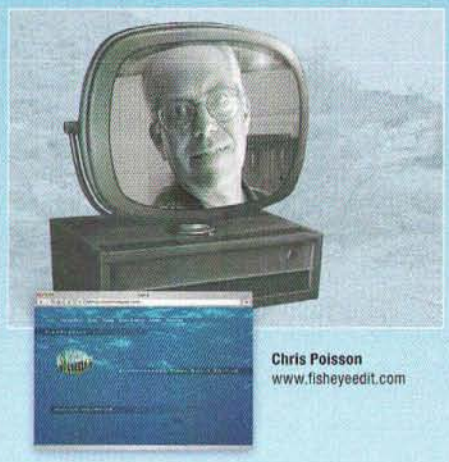


Ron Stanford
www.nicasurf.com

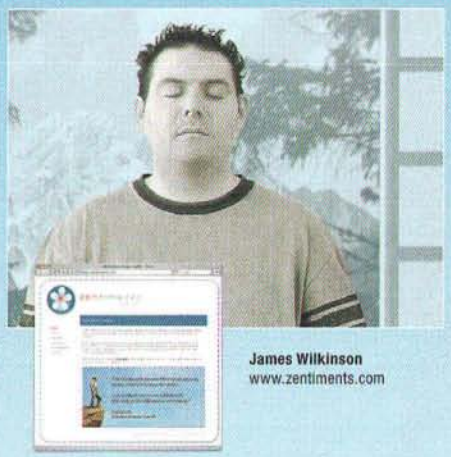
Blue Iceberg LLC
www.blue-iceberg.com



Jacqui Spadaro
www.lonelycustodian.org



Chris Poisson
www.fisheyeedit.com



James Wilkinson
www.zentiments.com

Free your mind, and design will follow.

Design the Web

The Web is no different from any other kind of design challenge. Attracting an audience, organizing information, communicating a message—these are your goals. Technicalities are important too, but not what you want to worry about every step of the way.

Let Your Creativity Flow

Most Web design tools—even so-called WYSIWYG editors—keep you mired in the details of code while you try to design, and unless you speak fluent HTML, that's not very helpful. Aside from stifling the creative process, it can make you reluctant to make changes—even when you know your efforts aren't quite right yet.

The Better Approach

Using Freeway, with its intuitive desktop-publishing interface, you simply draw the page the way you want it to look. Freeway's powerful Master Pages allow you to duplicate site elements and keep them up to date. Import graphics in almost any format or resolution, then scale, crop, rotate, and more, and let Freeway optimize all of your graphics on the fly!

Employ rollovers, rich media, and dynamic behaviors without hand-writing any code, because behind the scenes, Freeway is writing technically superb and efficient HTML code.

Viva la Difference!

We're so sure you'll love Freeway Pro and Freeway Express that we'll give you a full, unlimited version to use for 30 days, free! Once you experience Web design the way it should be, you'll never understand why you struggled with lesser tools. You may even fall in love with design—again.

Winner MacUser Awards 2003

Freeway Pro
Web Site Design and Publishing

Macworld AWARDS 2004 WINNER

Freeway Express
Editor's Choice: Best Consumer Software

Recent Reviews:
MacDesign: 5 out of 5
MacFormat: 5 out of 5
MacDirectory: 4.5 out of 5

freeway

www.softpress.com/mt

Figure 8 shows the layout of the **Record List** panel view. Data retrieved from the Record List sub-block are displayed in the three topmost NSTextField objects. Those retrieved from the Record Entry block are displayed in the NSTableView object. The controller assigned to this view is PDBRecList.

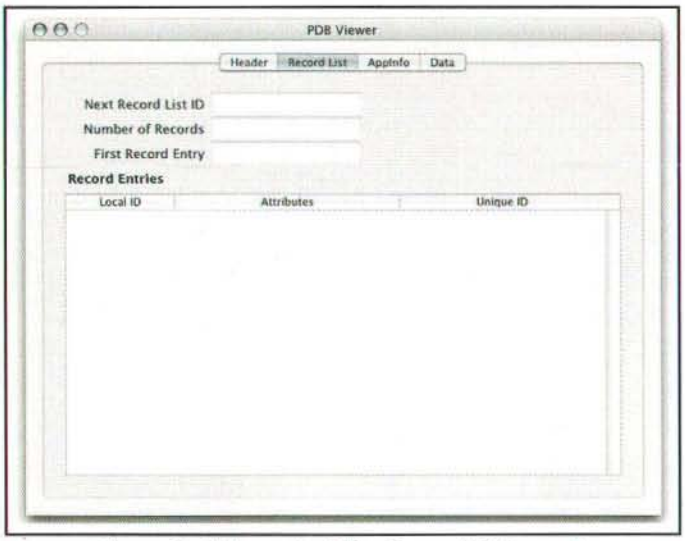


Figure 8. UI layout of the Record List panel.

The layout of the **AppInfo** panel view is shown in Figure 9. Integer data retrieved from the category sub-block of the AppInfo block is displayed in the two NSTextField objects. The category labels and their corresponding unique IDs are displayed in the NSTableView. Any application-specific data read from the AppInfo block is displayed in the NSTextView at the bottom. The controller assigned to this panel is PDBAppInfo.

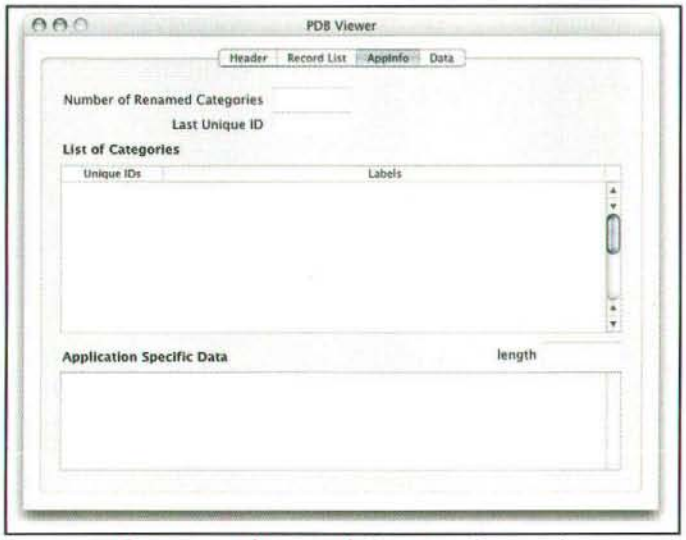


Figure 9. UI layout of the AppInfo panel.

Finally, Figure 10 shows the layout of the **Data** panel view. Application-specific data read from the SortInfo block would be displayed in the NSTextView at the bottom. The raw record data from the PDB file would be displayed in the NSTableView together with the corresponding local IDs.

Two controllers are assigned to update this panel view. PDBSortInfo handles the SortInfo data whereas PDBRecData handles the raw record data.

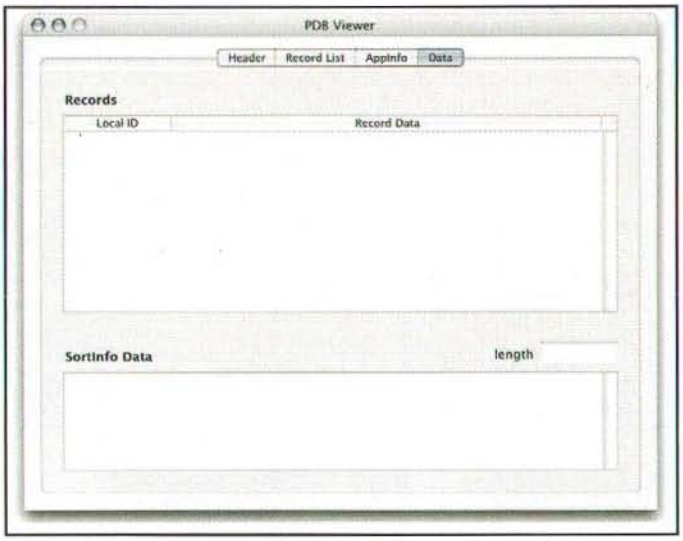


Figure 10. UI layout of the Data panel.

The Controllers

A total of 11 controllers comprise the PDB Viewer application. The five controllers mentioned in the preceding section maintain the display of information in each of the four tab panel views. Three controllers are used for data formatting. As for the remaining three, one maintains a PDB data buffer, another handles file I/O, and the last one coordinates the signal traffic between previous 10 controllers.

For purposes of length, I will only list the contents of the header files of each controller class. To see both the header and source files for each controller class, download the entire XCode project of the PDB Viewer from the MacTech web site (http://www.mactech.com).

The Main Controller

The PDBMain controller (Listing 12) coordinates all the signal traffic between the other controllers in the PDB Viewer application. It also intercepts any **File** menu selections and then invokes the appropriate controllers in the appropriate order.

Listing 12. The PDBMain controller (PDBMain.h).

```
#import "PDBFileIO.h"
#import "PDBHeader.h"
#import "PDBRecList.h"
#import "PDBAppInfo.h"
#import "PDBSortInfo.h"
#import "PDBRecData.h"

@interface PDBMain : NSObject
{
    // public instance outlets
    IBOutlet NSTabView    *pdbPanel;

    IBOutlet PDBFileIO    *pdbFile;
    IBOutlet PDBHeader    *pdbHeader;
    IBOutlet PDBRecList    *pdbRecList;
    IBOutlet PDBAppInfo    *pdbAppInfo;
    IBOutlet PDBSortInfo  *pdbSortInfo;
    IBOutlet PDBRecData    *pdbRecData;
}
```

```
// public action methods
- (IBAction)openDoc:(id)sender;
- (IBAction)saveDoc:(id)sender;

// protected accessor methods
- (void) updateViews;
- (void) setPList;
@end
```

To illustrate the role of PDBMain as a traffic coordinator, I prepared two signal diagrams showing the signal traffic during file-input and file-output. I used different line colors to help differentiate between independent signal flows.

Figure 11 is the signal traffic inside the PDB Viewer when the user selects a PDB file to be viewed by choosing the Open PDB menu option from the File menu. PDBMain receives the signal from the File menu via its openDoc action method. It then calls selectInput from PDBFileIO to start the file selection process.

Once a PDB file has been selected and read, PDBFileIO calls setBuffer from the PDBBuffer to archive the PDB data. PDBFileIO returns control to PDBMain which then calls the updateView methods of each of the following controllers: PDBHeader: PDBRecList, PDBAppInfo, PDBSortInfo, and PDBRecData. Each of these five controllers get their respective data blocks from PDBBuffer via its getBytesAt methods. They then process and display the PDB information in the appropriate UI outlets.

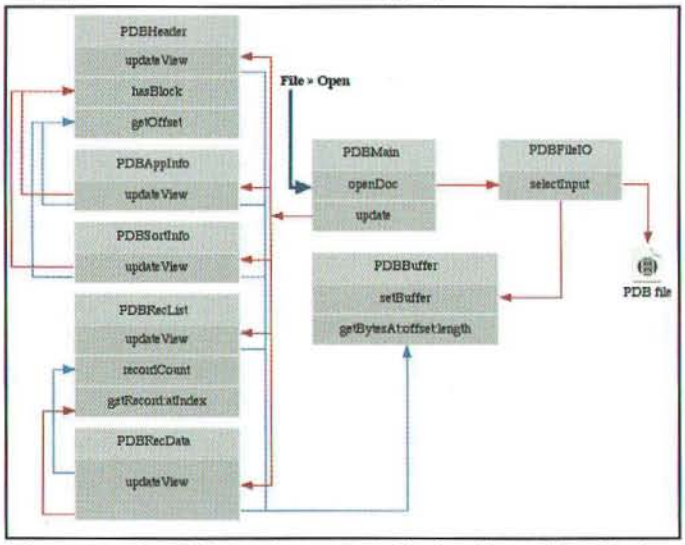


Figure 11. Signal flow when a PDB file is selected for display.

Figure 12 is the signal traffic inside the PDB Viewer when the user exports the PDB data into a plist file by choosing the Save .plist As option from the File menu. PDBMain receives the signal from the File menu via its saveDoc method. It then calls the setPList methods of the following controllers: PDBHeader, PDBRecList, PDBAppInfo and PDBRecData.

These five controllers then encapsulate their respective data blocks as NSDictionary objects. They send their dictionary objects to PDBBuffer via its setPList:forKey method. Once PDBMain regains control, it calls the selectOutput method of PDBFileIO to start the save process.

PDBFileIO prompts the user for a plist filename and destination directory. It then retrieves the consolidated NSDictionary object from PDBFileIO via its getPList method. Finally, PDBFileIO invokes the writeToFile method of the NSDictionary object to save its data into a plist file.

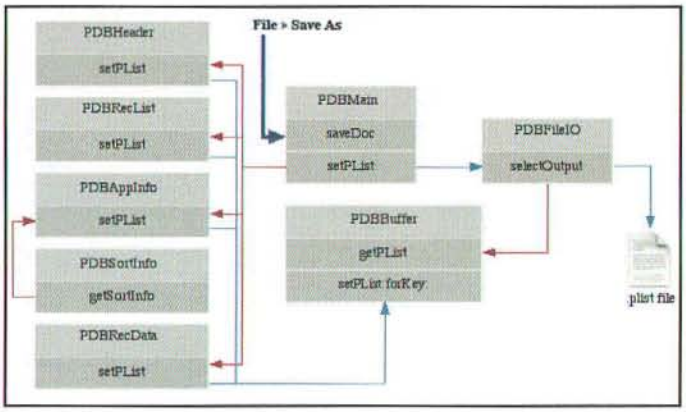


Figure 12. Signal flow when the PDB data is being saved to a plist file.

Data I/O and Buffering

The PDBFileIO controller (Listing 13) handles all the data input and output operations. This controller prompts the user for the PDB file to be viewed. It also prompts the user for a plist file name and destination directory. It reads the PDB data and submits it to the PDBBuffer controller for storage. It also queries the PDBBuffer for the NSDictionary object to be saved as a plist file.

Listing 13. The PDBFileIO controller (PDBFileIO.h).

```
#import "PDBBuffer.h"

// define the following private constants
#define PDB_nameWithExtension       YES
#define PDB_nameOnly                 NO

@interface PDBFileIO : NSObject
{
   // public instance outlets
      IBOutlet PDBBuffer   *pdbBuffer;

   // protected instance properties
      NSString             *srcPath;
}

// public accessor methods
- (void) selectInput;
- (void) selectOutput;

- (NSString *) getName:(BOOL)withExt;
@end
```

The PDBBuffer controller (Listing 14) manages the data retrieved from a selected PDB file. The controllers for each tab panel retrieve their respective data blocks through the getBytesAt method of the PDBBuffer.

PDBBuffer also manages an NSDictionary object that becomes the basis for the plist representation of the PDB data. The controllers for each tab panel submit their respective NSDictionary objects with the appropriate key values to the PDBBuffer using the setPlist:forKey method. The PDBBuffer then consolidates the NSDictionary objects into a single NSDictionary object for exportation.

Listing 14. The PDBBuffer controller (PDBBuffer.h).

```
@interface PDBBuffer : NSObject
{
   // protected instance properties
      NSMutableData       *dataBuffer;
      NSMutableDictionary  *dataDict;
}

// public accessor methods
- (NSData *) getBuffer;
- (NSData *) getDataAt:(unsigned int)offset
    length:(unsigned int)length;
- (NSDictionary *) getPList;

- (void) setBuffer:(NSData *)fileData;
- (unsigned char *) getBytesAt:(unsigned int)offset
    length:(unsigned int)length;
- (unsigned int) getBufferSize;

// public modifier methods
- (BOOL) setPList:(id)pdbData forKey:(id)pdbKey;
@end
```

Displaying PDB Information

As mentioned previously, there are 5 controllers that handle the display of data for each tab panel view. These controllers obtain their respective data block by querying PDBBuffer. These controllers also encapsulate their data as NSDictionary objects. They then send their NSDictionary objects to PDBBuffer to be consolidated for exportation.

The PDBHeader controller (Listing 15) handles the processing of data contained in the PDB header block. It displays the processed data in the Header tab panel of the PDB Viewer. The controller also calculates the time offsets needed to correctly display the PDB file's creation, modification and last-backup dates. This correction is necessary because to the different reference years used by the PalmOS and MacOS X.

Listing 15. The PDBHeader controller (PDBHeader.h).

```
#import "PDB.h"
#import "PDBBuffer.h"
#import "PDBFormatSig.h"

@interface PDBHeader : NSObject
{
   // protected instance outlets
      IBOutlet PDBBuffer       *dataSource;
      IBOutlet PDBFormatSig  *dataFormat;

      IBOutlet NSTextField     *fieldAttributes;
      IBOutlet NSTextField     *fieldDateBackup;
      IBOutlet NSTextField     *fieldDateCreate;
      IBOutlet NSTextField     *fieldDateModified;
      IBOutlet NSTextField     *fieldIDAppInfo;
      IBOutlet NSTextField     *fieldIDModified;
      IBOutlet NSTextField     *fieldIDSortInfo;
      IBOutlet NSTextField     *fieldIDUnique;
      IBOutlet NSTextField     *fieldName;
      IBOutlet NSTextField     *fieldSigCreator;
      IBOutlet NSTextField     *fieldSigType;
      IBOutlet NSTextField     *fieldVersion;

   // protected instance property
      NSData          *dataBuffer;
      DatabaseHdrType *dataPointer;
}

// public modifier methods
- (void) updateView;
- (void) showData;
- (void) setPList;
- (NSDate *) adjustDate:(unsigned long)pdbDate;

// public accessor methods
- (unsigned int) getOffset:(PDBBlockTypes)block;
- (BOOL) hasBlock:(PDBBlockTypes)block;
- (BOOL) getData;
@end
```

The PDBRecList controller (Listing 16) handles the processing of data contained in the PDB Record List and Record Entry blocks. It displays the processed data in the Record List tab panel of the PDB Viewer. Since that panel happens to have an NSTableView object, the PDBRecList controller also serves as the table's data source. Finally, the controller populates the First Record Entry field if and only if there is at least one valid record in the PDB file.

Listing 16. The PDBRecList controller (PDBRecList.h).

```
#import "PDB.h"
#import "PDBBuffer.h"
#import "PDBFormatHex.h"

@interface PDBRecList : NSObject
{
   // protected instance outlets
      IBOutlet PDBBuffer    *dataSource;
      IBOutlet PDBFormatHex   *dataFormat;

      IBOutlet NSTextField *fieldIDLocal;
      IBOutlet NSTextField *fieldRecordCount;
      IBOutlet NSTextField *fieldRecordFirst;
      IBOutlet NSTableView *tableEntries;

   // protected instance properties
      NSData       *dataBuffer;
      NSMutableArray *dataRecords;
      RecordListType *dataPointer;
}

// public modifier methods
- (void) updateView;
- (void) showRecordList;

// public accessor methods;
- (BOOL) hasEntries;
- (BOOL) hasRecords;
- (BOOL) getRecordList;
- (BOOL) getRecordEntries;

- (unsigned int) recordCount;
- (unsigned int) firstRecord;
- (unsigned int) getRecordAtIndex:(unsigned int)index;
- (void) setPList;
@end
```

The PDBAppInfo controller (Listing 17) handles the processing of data contained in the AppInfo block. It displays the processed data in the AppInfo tab panel of the PDB Viewer. Like the PDBRecList controller, it serves as a data source for the NSTableView object in the AppInfo panel. The controller also locates any application-specific data contained in the AppInfo data block. If found, the controller then displays the application-specific data in the appropriate NSTextView object using the PDBFormatData controller to reformat the data.

It should be noted that the PDBAppInfo controller only performs its data process if and only if the PDB data does include an AppInfo block.

Listing 17. The PDBAppInfo controller (PDBAppInfo.h).

```
#import "PDB.h"
#import "PDBBuffer.h"
#import "PDBHeader.h"
#import "PDBRecList.h"
#import "PDBFormatData.h"
#import "PDBSortInfo.h"

@interface PDBAppInfo : NSObject
{
   // protected instance outlets
      IBOutlet PDBBuffer       *dataSource;
      IBOutlet PDBHeader       *dataHeader;
      IBOutlet PDBRecList    *dataRecList;
      IBOutlet PDBFormatData   *dataStream;
      IBOutlet PDBSortInfo     *dataSortInfo;

      IBOutlet NSTextView  *fieldAppInfo;
      IBOutlet NSTextField   *fieldLastUniqueID;
      IBOutlet NSTextField   *fieldRenamedCount;
      IBOutlet NSTextField   *fieldDataLength;
      IBOutlet NSTableView *tableCategories;

   // protected instance properties
      NSData       *dataBuffer, *dataSpecific;
      AppInfoType *dataPointer;
}

// public modifier methods
- (void) updateView;
- (void) showDataInfo;
- (void) showDataSpecific;
- (void) setPList;

// public accessor methods
- (BOOL) getDataInfo;
- (BOOL) getDataSpecific;
@end
```

The PDBSortInfo (Listing 18) controller handles the processing of data contained in the SortInfo block. It displays the processed data in the NSTextView object located in the Data tab panel of the PDB Viewer. Like the PDBAppInfo controller, it only performs its data process if and only if the PDB data does include a SortInfo block. The controller also makes use of the PDBFormatData controller to reformat the data in human-readable form.

Listing 18. The PDBSortInfo controller (PDBSortInfo.h).

```
#import "PDB.h"
#import "PDBBuffer.h"
#import "PDBHeader.h"
#import "PDBRecList.h"
#import "PDBFormatData.h"

@interface PDBSortInfo : NSObject
{
   // protected instance outlets
      IBOutlet PDBBuffer       *dataSource;
      IBOutlet PDBHeader       *dataHeader;
      IBOutlet PDBRecList    *dataRecList;
      IBOutlet PDBFormatData   *dataStream;

      IBOutlet NSTextView  *fieldSortInfo;
      IBOutlet NSTextField   *fieldDataLength;

   // protected instance properties
      NSData   *dataBuffer;
}

// public modifier methods
- (void) updateView;

// public accessor methods
- (BOOL) getData;
- (NSData *) getSortInfo;
@end
```

The PDBRecData controller (Listing 19) handles the processing of any existing raw record data contained in the PDB file. It determines the location and length of each raw record using the information provided by the PDBRecList controller. It then parses out these records and displays them in the NSTableView object located on the Data panel of the PDB Viewer.

Listing 19. The PDBRecData controller (PDBRecData.h)

```
#import "PDB.h"
#import "PDBBuffer.h"
#import "PDBRecList.h"
#import "PDBFormatHex.h"
#import "PDBFormatData.h"

@interface PDBRecData : NSObject
{
   // protected instance outlets
      IBOutlet PDBBuffer    *dataSource;
      IBOutlet PDBRecList     *dataRecList;
      IBOutlet PDBFormatHex   *dataFormat;
      IBOutlet PDBFormatData  *dataStream;

      IBOutlet NSTableView  *tableRecords;

   // protected instance properties
      NSMutableArray    *dataRecords;
}

// public modifier methods
- (void) updateView;
- (void) setPList;

// public accessor methods
- (BOOL) getData;
@end
```

Data Formatting

There are three controllers subclassed from the NSFormatter object.. First of these is PDBFormatSig (Listing 20). This controller takes the original integer values of the type and creator signatures of the PDB file and converts them into the familiar four-character signatures.

Two of the NSTextFields in the Header tab panel uses PDBFormatSig as their formatter. PDBHeader also uses PDBFormatSig to format the type/creator signatures when preparing the PDB header data for output to a plist file.

Listing 20. The PDBFormatSig controller (PDBFormatSig.h)

```
@interface PDBFormatSig : NSFormatter
{
}

// public modifier methods
- (NSString *) int2sig:(unsigned int)intArg;
@end
```

The second controller, PDBFormatHex (Listing 21), takes an unsigned integer value and generates the corresponding hexadecimal string.

Three NSTextFields in the Data tab panel use PDBFormatHex as their formatter. These three fields display the

attributes, AppInfo and SortInfo IDs stored in the PDB header block. The NSTextField used to display the nextRecordID value in the Record List tab panel also uses PDBFormatHex as its formatter. Finally, the NSTableViews in both the Record List and Data tab panels use PDBFormatHex to format the values displayed in their Local ID columns.

Listing 21. The PDBFormatHex controller (PDBFormatHex.h)

```
@interface PDBFormatHex : NSFormatter
{
}

// public modifier methods
- (NSString *) int2hex:(unsigned int)intArg;
@end
```

The third controller, PDBFormatData (Listing 22), takes an NSData object and converts each byte value into some human-readable form. A byte value of 0x00 is represented by a bullet (•). Byte values within the range of 0x20 and 0x7e are represented by their equivalent ASCII character. Any other byte values are represented by an ellipsis (...).

The PDBFormatData is used by the PDBAppInfo and PDBSortInfo controllers to format any application-specific data prior to being displayed in their respective NSTextViews. PDBRecData controller also uses PDBFormatData to format any raw record data prior to being displayed in the Record Data column of the NSTableView object.

Listing 22. The PDBFormatData controller (PDBFormatData.h)

```
@interface PDBFormatData : NSFormatter
{
}

// public modifier methods
- (NSString *)data2stream:(NSData *)dataArg;
@end
```

Possible Improvements

For the purposes of this article, the PDB Viewer only allows me to load and view the data contained inside a PDB file as well as export it to a plist file. There are however, a number of directions I can pursue that would further improve upon the application's usability. Some of the more notable ones are as follows:

- Add the ability to load and view a plist file containing PDB data and generate a PDB file from the data read. To accomplish this, PDBFileIO needs to correctly recognize that the data being read is coming from a plist file. PDBBuffer will then separate the data into individual blocks (Header, Record List, etc..) and store each block as an NSDictionary entry. Then PDBHeader and its kind will then asks for their respective data blocks to be parsed and displayed accordingly.

- Use a hex-editor style UI to display any application-specific data from the AppInfo and SortInfo data blocks. One way to accomplish this is to subclass NSTableView. The subclass (let's call it PDBHexTable) would consist of 18 columns: one for the data offsets, one for the ASCII stream and the rest for the hexadecimal values. Also, a controller (PDBHexShow) would be designed to handle the display of data in the PDBHexTable.

- Allow on-screen editing of PDB data and then save the edited data back into a separate PDB or plist file. The controllers for each of the tab panel views (such as PDBHeader) will have to be modified to support this feature. Also, PDBFileIO will have to be modified to allow users to select the type of file into which the data would be saved.

Summary

I have shown you the basic structure of a PDB file and how information is stored in that file. I have also demonstrated how to use XCode and Cocoa to build a basic PDB Viewer that allows us to view the contents of a PDB file. I was also able to add an additional feature to the PDB Viewer thus allowing me to export the PDB data into a plist file for later viewing and perhaps even editing.

Once again, the complete XCode project for the PDB Viewer is available for downloading at the MacTech website (www.mactech.com).

In my next article, I will cover the basic structure of a Palm Resource file (or PRC). Learning the data structure of the PRC file is an important step towards learning PalmOS development as this is the universal executable format used by a PalmOS application. I will show how to develop a PRC Viewer that would allow me to view the contents of a PRC file as well as save the PRC data into a plist file for later viewing/editing. In addition, I will also show how to allow the PRC Viewer to load the contents of a plist file (which contains PRC data) and then recreate a PRC file.

Bibliography and References

Stone, Denise. "Palm OS ® File Formats". *Exploring Palm OS ®*. Document Number 3120-002. 2004 Nov 9. pp. 4 – 12, 14 – 19. PalmSource, Inc.

Wilson, Greg; Ostrem, Jean; Rey, Christopher. *Palm OS Programmers ® Companion, Volume 1*. Document Number 3004-008. 2003 Sept 4. PalmSource, Inc.

Rhodes, Neil; McKeehan, Julie. *Palm Programming: The Developer's Guide*. Copyright 1999. O'Reilly & Associates.

About The Author

JC is a freelance engineering consultant currently residing in North Vancouver, BC. He divides his time between custom application development for OS X, technical writing and teaching origami to children at the local district libraries.

Building An AppleScript-Based Automator Action

There has been a lot of excitement in the developer community around the release of Mac OS X 10.4. Unique technologies like Automator, Dashboard, and Spotlight are providing new opportunities for Mac developers to build unique tools that appeal to users everywhere. This month, we are going to walk through the process of developing for one of these great new technologies, Automator.

Automator is Apple's new technology that is helping users everywhere to begin automating repetitive and time consuming tasks in their own unique workflows, allowing them to become more efficient. By placing actions, which are single automated tasks, together in a sequence, users are able to construct a fully automated workflow, without the need to write a single line of code. How does this help us, as developers, you may be asking? Well, someone needs to create the actions that give users this power.

Automator actions are built in Xcode, and are typically developed using either Objective-C, AppleScript, or a combination of these, and possibly other languages. Objective-C may be used to develop actions that interact with core components of the OS, or applications with a public API. AppleScript may be used to develop actions that interact with any scriptable application or process on the Mac.

In this article, we will walk through the process of creating a basic Automator action with AppleScript. This brief overview should provide you with a basic understanding of the primary steps involved in creating a simple AppleScript-based action. Once you understand the basic concepts of building an action, then you can take additional steps on your own to begin expanding your knowledge through additional resources. Before long, you will be creating complex actions that interact with a variety of scriptable applications or processes, and sharing those actions with users. Obviously, it should go without saying that you will need Mac OS X 10.4 and Xcode 2 or higher to create an Automator action.

Getting Started

There are a number of steps involved in the creation of an AppleScript-based Automator action, and we will move through the process fairly quickly.

The first step in creating an action is to determine what the action will do. I've done this part for you already. Our action will accept a list of values as input, and then display a choose from list dialog to the user, allowing the user to make a selection. The value specified by the user will then be passed as output on to the next action in an Automator workflow sequence. An action such as this might be used to give a user the opportunity to process only a specified set of data during execution of a workflow.

> The choose from list command is found in the ***User Interaction*** suite in the ***Standard Additions*** scripting addition, included with Mac OS X.

Create an Action Project

Once you have determined what your action will do, the next step is to create a new Xcode project. To do this, launch Xcode and select ***New Project...*** from the ***File*** menu. You will then be

prompted to select from a list of pre-existing project templates. Apple has included a handful of Automator action templates with Xcode to get you started. Select the *AppleScript Automator Action* template, and click the *Next* button to proceed. See figure 1.

Figure 1. Choosing an Automator Action Project Template

If you are feeling adventurous, or if you prefer developing in other languages, be sure to check out the *Cocoa Automator Action* and *Shell Script Automator Action* (Xcode 2.1 or higher) templates, also included with Xcode. Also, be sure to check out the example Automator action projects, complete with sample code, located in the Developer > Examples > Automator folder.

Next, specify a name for your Automator action, in this case, *Choose List Items*. Specify a directory for the project, and click the *Finish* button to create the action project. See figure 2.

Figure 2. Specifying a Project Name and Directory

You should now have an Automator action project opened in front of you in Xcode. See figure 3. If you are new to Xcode, then the project may look a little overwhelming to you at first glance. However, there are really only a handful of components with which we, as AppleScript developers, will need to interact. These components are:

main.applescript – This is your action's main AppleScript file. It will contain the AppleScript code that will trigger when the action is run in an Automator workflow.

main.nib – This is your action's interface, as it will be displayed when the action is placed into the workflow view in Automator's interface.

info.plist – This is essentially a configuration file. It will indicate how your action should be handled within Automator, and within a workflow sequence.

An English instance of an *InfoPlist.strings* file is also present, and may be used to specify English translations of properties contained within the *info.plist* file. Optionally, additional versions of this file may be added for additional translations. For this sample action, we will not use the *InfoPlist.strings* file.

We will discuss each of these components in greater detail as we build our action project.

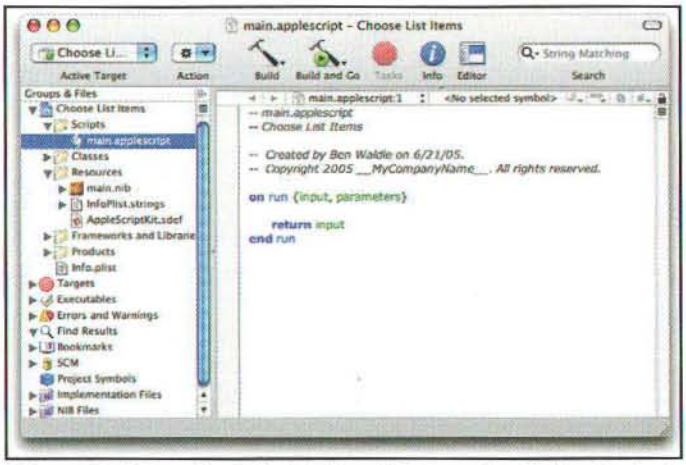


Figure 3. A New AppleScript-Based Automator Action Project

Update the Action's Properties

Once we have created our action project, we need to make some adjustments to the *info.plist* file within the action project. As previously mentioned, the *info.plist* file is an XML file, which provides information about the action to the Automator application. There are a number of properties and values included in the *info.plist* file. For the purposes of the action we

are building, we will discuss only a handful. At this time, take care not to make changes to any properties included in this file, which are not specified below.

There are a number of ways that you can edit an action's *info.plist* file. For this project, we will edit the file's XML code directly. Click on the *info.plist* file in your action's project to view the file's contents within your Xcode window. Skim the *info.plist* file, paying special attention to the properties listed below, and making any adjustments indicated.

AMAccepts

This property provides Automator with information about the input that an action will accept. This property contains an XML dictionary, which specifies whether the input for the action is optional, and which types of input are acceptable. If an action is configured to accept a certain type of input, then Automator will generate an error if an incompatible input type is passed to the action.

An action may be configured to accept multiple input types, if desired, and each input type must be specified as a ***universal type identifier*** (UTI). A list of valid UTI's is included with the Automator developer documentation. For our action, we will simply make use of the default UTI for this property, com.apple.applescript.object, which indicates a generic AppleScript object. Because this is configured by default, you should not need to modify any aspects of this property. The property should appear as follows within your *info.plist* file:

```
<key>AMAccepts</key>
<dict>
  <key>Container</key>
  <string>List</string>
  <key>Optional</key>
  <false/>
  <key>Types</key>
  <array>
    <string>com.apple.applescript.object</string>
  </array>
</dict>
```

AMApplication

This property specifies the category in which the action will appear in the *Library* list within Automator's interface. For our action, set this property to a value of Automator. This will cause the action to appear within the *Automator* category.

```
<key>AMApplication</key>
<string>Automator</string>
```

AMCanShowSelectedItemsWhenRun and AMCanShowWhenRun

These properties indicate whether the user should be allowed to configure the action's interface to be displayed when run within a workflow. For our action, set the values of both of these properties to false, as we do not want the user to have the ability to display the action's interface during processing.

```
<key>AMCanShowSelectedItemsWhenRun</key>
<false/>
<key>AMCanShowWhenRun</key>
<false/>
```

AMCategory

This property is used to help Automator group similar actions together, internally. Currently, the value of this property is utilized by Automator only when the user performs a search via the search field in Automator's toolbar. For our action, set this property's value to Dialog.

```
<key>AMCategory</key>
<string>Dialog</string>
```

AMDefaultParameters

This property is used to link attributes of our action's interface to the action's code. We will revisit this property shortly, once we have created our action's interface, and we will specify its value at that time.

AMDescription

This property contains an XML dictionary, which contains the information that Automator displays in the description area when the action is selected. This value may be used to specify a variety of different types of information. For our action, we will use only a few. Configure this property in your action's *info.plist* file to match the following:

```
<key>AMDescription</key>
<dict>
  <key>AMDInput</key>
  <string>A list of values, which may be coerced to text
format</string>
  <key>AMDOptions</key>
  <string>Allow multiple selections; allow empty
selections</string>
  <key>AMDResult</key>
  <string>Specified list items</string>
  <key>AMDSummary</key>
  <string>This action will prompt the user to select
from a list of values.</string>
</dict>
```

AMIconName

This property is used to specify the name of a graphic file that will serve as the action's icon in Automator's *Action* list, as well as in the action's description, when the action is selected. You may specify the name of a graphic file within your project, within Automator's bundle, or within the "CoreTypes" bundle (found in *System > Library > CoreServices*). For our action, we will specify a graphic that is included within Automator's bundle, a standard AppleScript icon:

```
<key>AMIconName</key>
<string>AppleScriptLarge</string>
```

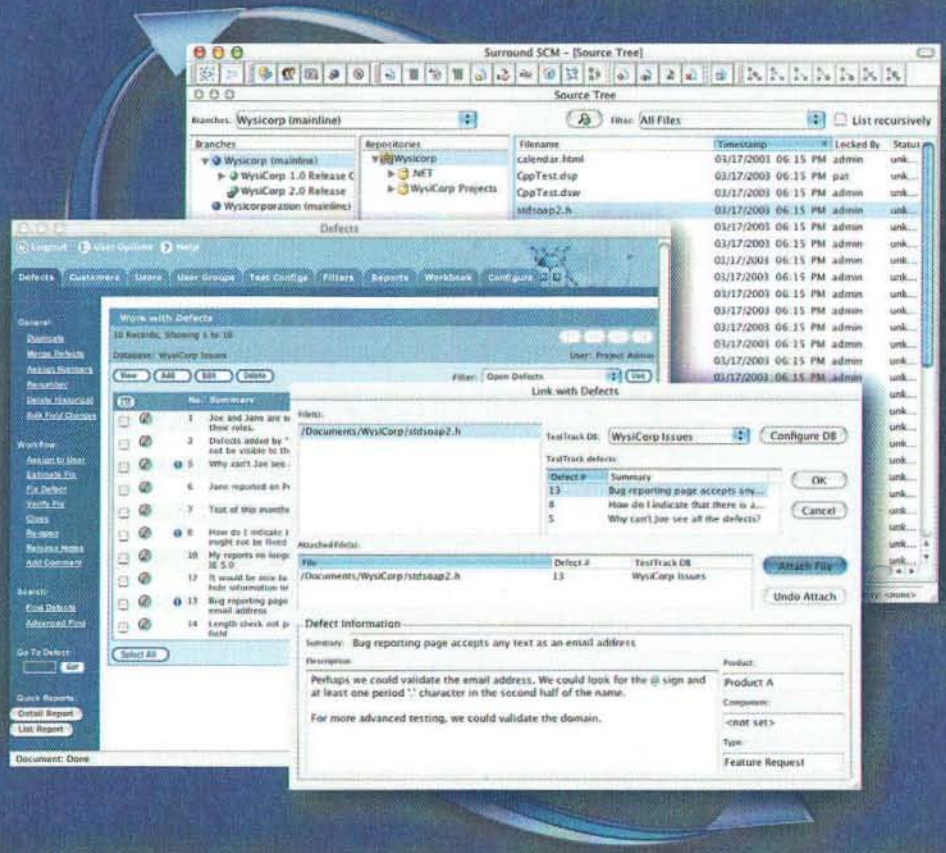
AMKeywords

This property is used to specify keywords for an action. These keywords will be accessed by Automator when a user performs a search via the search field in Automator's toolbar. For our action, we will specify select and display as keywords. Feel free to assign other keywords as well, if you would like.

```
<key>AMKeywords</key>
<array>
  <string>select</string>
  <string>display</string>
</array>
```

Complete Source Control and Defect Management for Mac OS X

Seapine Software™
changing the world of software development



Effective source code control and defect tracking require powerful, flexible, and easy-to-use tools—Surround SCM and TestTrack Pro

- Complete source code control with private workspaces, automatic merging, role-based security, and more
- Comprehensive defect management — track bug reports and change requests, define workflow, customize fields
- Fast and secure remote access to your source files and defects — work from anywhere
- Advanced branching simplifies managing multiple versions of your products
- Link code changes with defects and change requests — know who changed what, when, and why
- Scalable and reliable cross-platform, client/server solutions support Mac OS X, Windows, Linux, and Solaris
- Exchange data using XML and ODBC, extend and automate with SOAP support
- Licenses priced to fit your budget

Seapine Software Product Lifecycle Management
Award winning, easy-to-use software development tools

Seapine **Surround SCM**

Seapine **TestTrack PRO**

software development 13th annual product excellence award JOLT

Download Surround SCM and TestTrack Pro at www.seapine.com or call 1-888-683-6456

AMName

This property contains the name of your action, as it will appear in the ***Action*** list within Automator. This property should already be populated for you, but it still bears mentioning. It should appear as follows within your action's ***info.plist*** file.

```
<key>AMName</key>
<string>Choose List Items</string>
```

AMProvides

This property is similar to the AMAccepts property. It contains an XML dictionary, and indicates the type of output that the action will provide to the next action in a workflow sequence. Like the AMAccepts property, the output type must be a valid UTI. By default, this property should already be configured to output a generic AppleScript object, so you should not need to adjust it. This property should appear as follows within your action's ***info.plist*** file.

```
<key>AMProvides</key>
<dict>
  <key>Container</key>
  <string>List</string>
  <key>Types</key>
  <array>
    <string>com.apple.applescript.object</string>
  </array>
</dict>
```

Localized Strings

As previously mentioned, you may use an ***InfoPlist.strings*** file to specify localized versions of strings within your action's ***info.plist*** file. However, for the purposes of this example action, we will not perform this task. Therefore, click on the ***InfoPlist.strings*** file in your action project, displaying its contents in Xcode. Next, delete the existing text from this file, leaving the ***InfoPlist.strings*** file blank.

Build the Action's Interface

The next step in building our Automator action is to create our action's interface. First, double click on the ***main.nib*** file in your action project to open the action's nib within Interface Builder. Once opened, double click on the nib's view instance, if it is not already displayed for you.

Assuming that you already have a basic understanding of how to use Interface Builder, then you may begin adding interface elements to your action's view. For our action, add two checkboxes, set their size to small, and title them ***Allow Multiple Selections*** and ***Allow Empty Selections***, as shown in figure 4. These checkboxes will be the settings within our action's interface, which the user will be able to configure from within Automator.

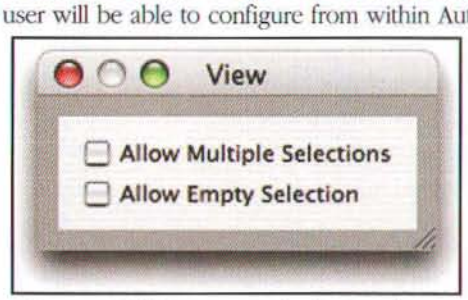


Figure 4. Editing an Action's Interface

For more complex actions, you may add additional interface elements. However, be sure to adhere to Apple's Aqua human interface design guidelines, taking into account the limited amount of space within Automator's interface.

Bind The Interface to the Action's Code

Once an action's interface has been designed, the interface elements must be linked to the action's code. This will allow the action to detect changes made to the action's settings by the user, and trigger the appropriate processing code.

There are multiple ways of linking interface elements to the code of an action. However, the most straightforward is to make use of ***Cocoa bindings***. This is the method that we will use for our action. The following steps will walk you through the process of creating these bindings.

Create Parameter Keys

The first step in binding interface elements to action code is to assign parameters. These parameters will later be attached to attributes of the interface elements, and inserted into our action's code. By doing this, changes made to the specified interface element attributes will automatically synchronize to the code of our action. Begin by clicking the ***Parameters*** instance in the action's nib. See figure 5.

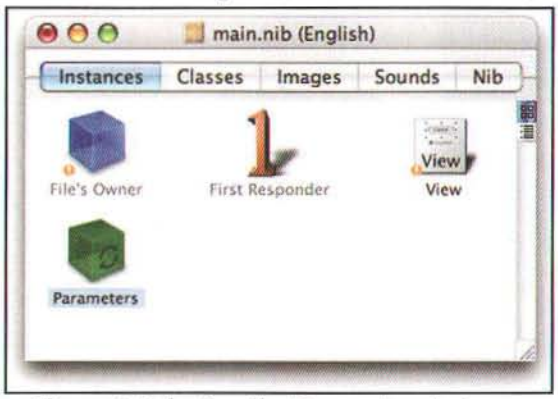


Figure 5. Selecting the Parameters Instance

Next, type ***command + 1***. This will open the ***Inspector*** panel, if not already opened, and display the ***Attributes*** pane for the selected ***Parameters*** instance. Next, click the ***Add*** button in the ***Attributes*** pane of the ***Inspector*** panel, and add two keys. As the new keys are created, double click on each of them, and re-name them ***allowEmptySelection*** and ***allowMultipleSelections***, as shown in figure 6.

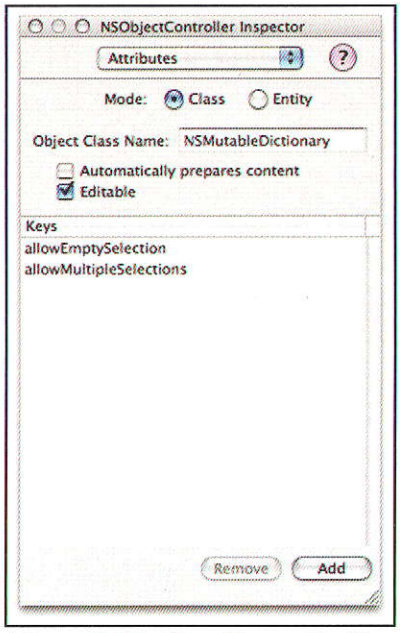


Figure 6. Interface Parameter Keys

Bind Parameter Keys to Interface Elements

Once you have created parameter keys, select each of the checkboxes in the action's window view, and perform the following steps. Type *command + 4* to display the *Bindings* pane in the *Inspector* panel for the current checkbox. Click on *value* in the *Bindings* panes of the *Inspector* panel, and select the parameter key that corresponds to the current checkbox in the *Model Key Path* field. See figure 7 for an example of a properly configured parameter binding for the *Allow Multiple Selections* checkbox.

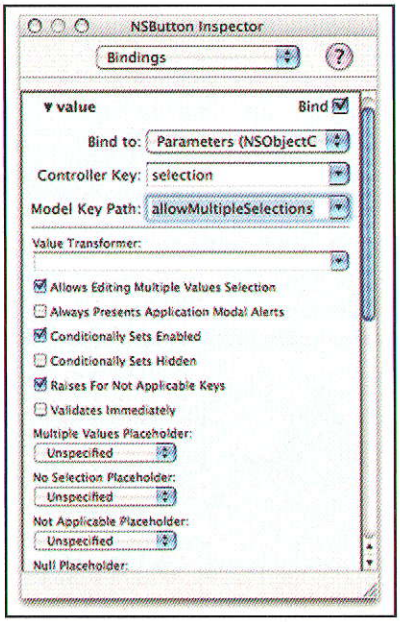


Figure 7. Binding Parameter Keys to Interface Element Attributes

Once parameter keys have been bound to the values of both checkboxes, save the action's interface in Interface Builder, and return to Xcode.

Specify Parameters in the info.plist File

Now that we have configured the bindings within our action's interface, we need to link them to our code, so that they will synchronize automatically when modified by the user. Click on the *info.plist* file again to display the list of properties for the action. Now, we will revisit the AMDefaultParameters property, which we mentioned briefly.

The AMDefaultParameters property should contain an XML dictionary, containing key/value combinations for the various parameter keys specified within your action's interface. Values specified for these keys will serve as default values for any bound interface elements. For our action, set the AMDefaultParameters property to contain a key for each of the parameter keys that we specified, along with a value of false for each. The properly configured property should appear as follows within your action's *info.plist* file:

```
<key>AMDefaultParameters</key>
<dict>
  <key>allowMultipleSelections</key>
  <false/>
  <key>allowEmptySelection</key>
  <false/>
</dict>
```

Write the Action's Code

Next, we are finally ready to begin writing the processing code for our action. We will be doing this entirely with AppleScript. Click on the *main.applescript* file to display the action's AppleScript code within Xcode. Next, specify the following code for the action:

```
on run {input, parameters}
  if (class of input) is not equal to list then set
input to {input}
  if input = {} then return input

  set inputStrings to {}
  repeat with a from 1 to length of input
    set end of inputStrings to item a of input as string
  end repeat

  set allowMultipleSelections to
|allowMultipleSelections| of parameters
  set allowEmptySelection to |allowEmptySelection| of
parameters

  set outputStrings to choose from list inputStrings
multiple selections allowed allowMultipleSelections empty
selection allowed allowEmptySelection
  if outputStrings = false then error number -128

  set output to {}
  repeat with a from 1 to length of inputStrings
    if outputStrings contains (item a of inputStrings)
then set end of output to item a of input
  end repeat

  return output
end run
```

As you write processing code for an action, take care to add protection for scenarios that might cause the action to generate

an error during processing. For example, our action expects a list of values to be provided as input. Therefore, I have added code to ensure that the specified value is a list, coercing it to a list if necessary.

Test the Action

Once the action's processing code has been written, you are ready to begin testing your action. You may do so by selecting ***Build and Run*** from the ***Build*** menu within Xcode. Doing so will launch a temporary instance of the Automator application, and your action should be accessible for testing.

To test our specific action, construct a sample workflow. This workflow may consist of a ***Run AppleScript*** action, our ***Choose List Items*** action, and a ***View Results*** action. See figure 8.

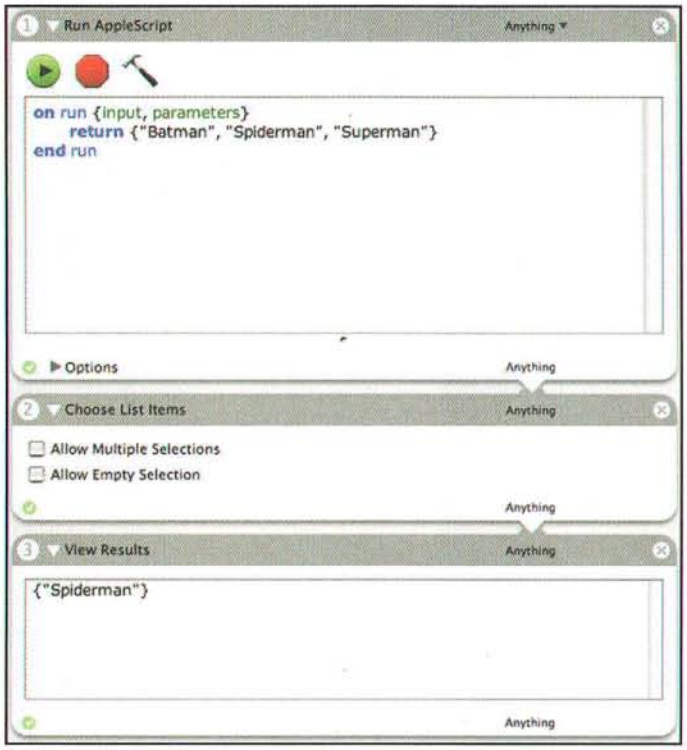


Figure 8. An Example of a Test Workflow

If all goes well, your action should display a list of passed input values when run within a workflow, and output the user-specified values. Inevitably, you may make a mistake, which could prevent your action from appearing within Automator, or from running properly within a workflow sequence. It happens to the best of us. To help resolve any issues that may occur during development of this action, you may download and consult the source code for this example action from the following URL:

<http://www.automatedworkflows.com/files/demos/MacTECH.08.05.Example.zip>.

In Closing

This article should serve as an initial guide to get you started with building your own AppleScript-based Automator actions. However, it is not meant to serve as a definitive guide to Automator development by any stretch of the imagination. For detailed information about creating Automator actions in AppleScript, as well as Objective-C, please refer to the developer documentation included with Xcode, and also available online via the Apple Developer Connection.

If you prefer a book on Automator, then be sure to check out my comprehensive ***Mac OS X Technology Guide to Automator***, available from SpiderWorks http://www.spiderworks.com in both print and eBook formats. The first section of the book covers using Automator, and the second section covers developing your own custom actions. If you think that you need a refresher on AppleScript itself, be sure to check out *Danny Goodman's AppleScript Handbook* while you're there as well. Sample chapters of both books are available for download.

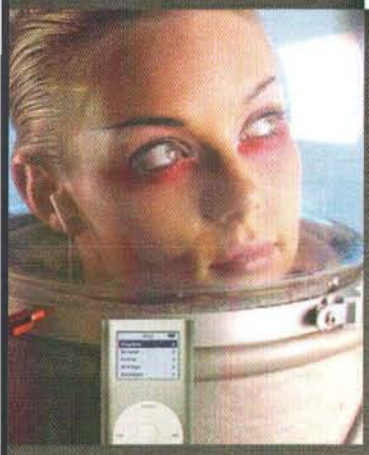
Until next time, keep scripting!

About The Author

Benjamin Waldie, author of the best selling eBooks "AppleScripting the Finder" and "Mac OS X Technology Guide to Automator", available exclusively from www.spiderworks.com, is president of Automated Workflows, LLC, a firm specializing in AppleScript and workflow automation consulting. For years, Benjamin has developed professional AppleScript-based solutions for businesses including Adobe Systems, Apple Computer, NASA, PC World, and TV Guide. In addition to his role as a consultant, Benjamin is an evangelist of AppleScript, and can frequently be seen presenting at Macintosh User Groups, Macworld, and other events. For additional information about Benjamin, please visit www.automatedworkflows.com, or email Benjamin at applescriptguru@mac.com.

$32 < 1 Year

2 Years > $64

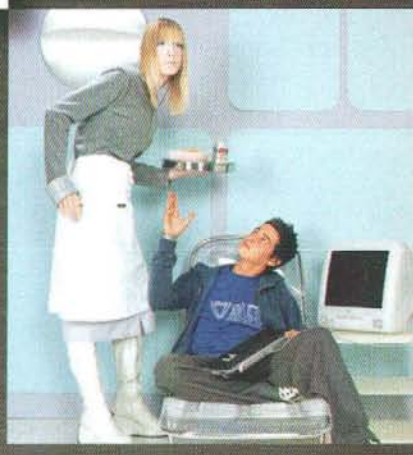


Interviews

Tapping into the world of celebrities and their Macs, only MacDirectory offers exclusive interviews. Get a close and personal view from Sarah Jessica Parker, Steve Jobs, Madonna, Harry Connick Jr., George Lucas, Jennifer Jason Leigh, Steve Woz and other leaders in the Mac community.

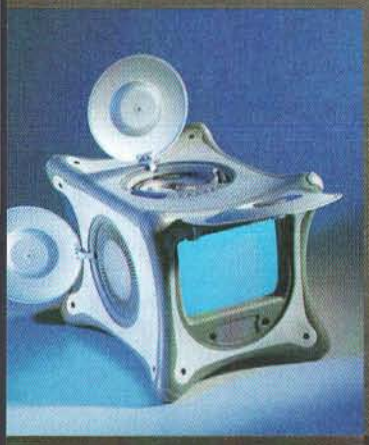
Features

Designers, writers, musicians, business leaders & our technical expert team offer their own personal interpretation of things that only the Mac system can deliver. With more than 200 pages of news, insights, trends and the largest Macintosh buyer's guide including over 5,000 Mac products and services.



MacDirectory

BEYOND ANY MACINTOSH MAGAZINE.

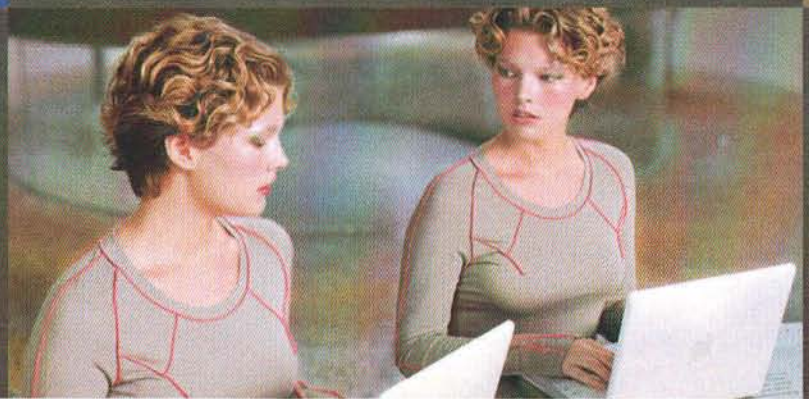


Culture

MacDirectory takes you to the wildest corners of the world and uncovers how Macintosh computers are being used by other cultures. Travel to Japan, Australia, Germany, Brazil & Russia and learn more about Apple's cultural impact around the globe.

Reviews

Find out all you need to know about the latest Mac products including the hottest Mac OS software and hardware.



Subscribe >

macdirectory.com

Send check or money order to:
MacDirectory Subscription Dept.
326 A Street, 2C
Boston, MA 02210

Mac OS X Server 10.4

Part Two Of Our Overview Of Mac OS X 10.4 Server

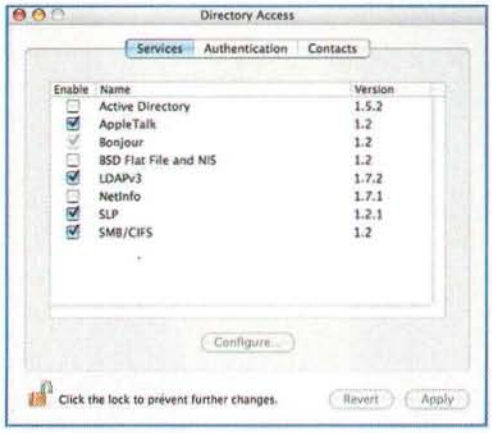
Last time, we took a long look at Mac OS X 10.4 Server from the Server Admin point of view, and concentrated mainly on server setup, features, and some management. Now, we're going to look at Mac OS X 10.4 Server from the Workgroup Manager point of view, or the tools and new features that Mac OS X 10.4 Server provides so you can better manage your network.

Workgroup Manager

Workgroup Manager is the tool you use to manage your overall Mac OS X network. Where Server Admin works with one server at a time, and only servers, Workgroup Manager works with multiple servers, client machines and client users. It's the primary tool for:

- Setting sharepoints and access policies for those sharepoints
- Setting client machines and client machine groups, their access policies and preferences
- Setting users and user groups, their access policies and preferences
- Manual manipulation of Open Directory data

Workgroup Manager is also the way you handle directory data from multiple directory systems. If you have multiple systems defined via the Directory Access application, then Workgroup Manager can work with those systems.



Directory Access in Mac OS X 10.4.x

One important point here is that you don't have to run Workgroup Manager on the server your using it with. You can also run Directory Access against a remote Mac OS X Server box via the cmd-K option, and this allows for what is called "directory mode", accessed via the "View Directories" option, or cmd-D in Workgroup Manager. In directory mode, you run Workgroup Manager from a remote administration Mac, that can be bound, via Directory Access to different directory systems than the server you're managing. This allows, for example, an Open Directory Master to integrate data from Active Directory

without having to be itself bound to Active Directory. This allows for a great deal of flexibility, although my personal experience with directory mode has been inconsistent.

This does bring to mind one annoyance with directory service handling in Mac OS X in general, and that is how dependent it is on search order. If the search order in Directory Access is incorrect, you're in a world of hurt. As well, from what I can tell, if you have two external directory systems defined, and you try to authenticate, and the first one can't authenticate you, the directory services code won't fall through to the next one, and you can't authenticate. This is just a little annoying.

However, when directory mode in Workgroup Manager is working, and you have your authentication paths set right, it's neat as heck to use.

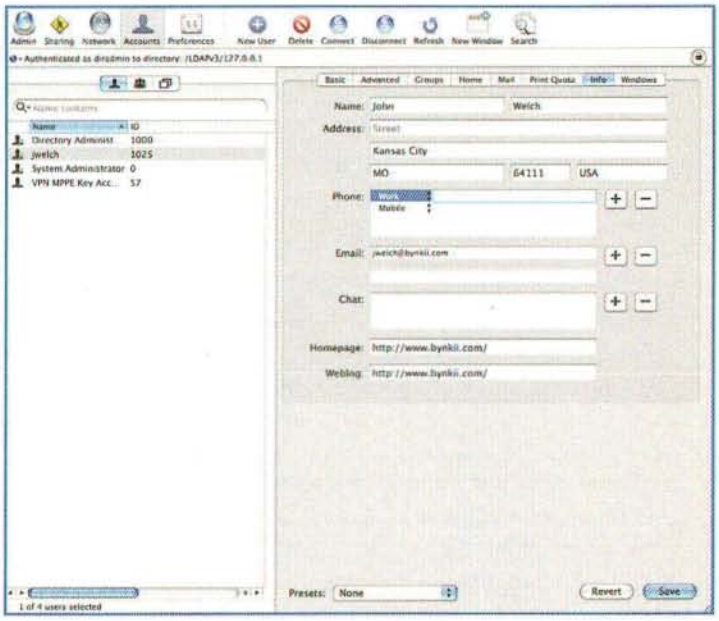
New Features

Since this ***is*** an article about Mac OS X 10.4 Server, I'm obviously going to concentrate on the new features that are in Mac OS X 10.4 Server that relate to Workgroup Manager. This isn't really a Workgroup Manager review; I'm just using that product as a framework. One change has to do with the new "Directory Admin" user concept. This just allows you to set up a Server so that the local machine administrator is not, by default the directory administrator, and in fact, the local administrator account that you first set up is a NetInfo only account, not an LDAP account. A second new feature is the ***Search*** button in the Workgroup Manager toolbar that allows you to search the Open Directory database by a number of items, such as UserID, Real Name, Comment, etc. If you have thousands of records in your Open Directory domain, that's a right handy feature to have.

User Accounts

The first obvious changes are due to the ACLs in Mac OS X 10.4. The old 16-group limits are gone, and there's support for inherited groups. As I said in July's article on this, ACLs give you great power, but you have to be careful, especially if you're talking about combined Active Directory and Open Directory networks. Careless application of ACLs will create security holes the likes of which you've never seen. Setting up groups for a user is still the same, although there's a new option for seeing inherited groups. This is for ACL usage, since you can have explicit and inherited permissions and groups. Mail and Print setup hasn't changed, although with the improvements to Mac OS X 10.4 Server's print architecture, the existing features probably work ***much*** better.

A new feature in Mac OS X 10.4 Server's client management is the ***Info*** tab for user accounts. This allows you to directly enter personal information on a user such as name, address, IM handle, etc. This is not a requirement for the account to work, but is a convenience for those using Open Directory as a shared address book. You could do this in Mac OS X 10.3 Server, but you had to directly edit the directory information, and it didn't always work correctly.



Workgroup Manager's Info Tab

The Windows settings for User accounts is unchanged, and the Inspector tab has only minor changes, such as showing the size of each entry in the user's record. One thing that hasn't changed from Mac OS X 10.3, and I really wish would, is the pane management in Workgroup Manager. You can't resize the panes in Workgroup Manager to show more info. You need to see the full Generated UID number? Changing the size of the window only increases the size of the account listing on the left. The data display on the right cannot be grown or shrunk depending on your needs, so get to scrolling. Shades of Windows 3.X! For a company that gets UI right far more than most, when they get it wrong, it's doubly frustrating.

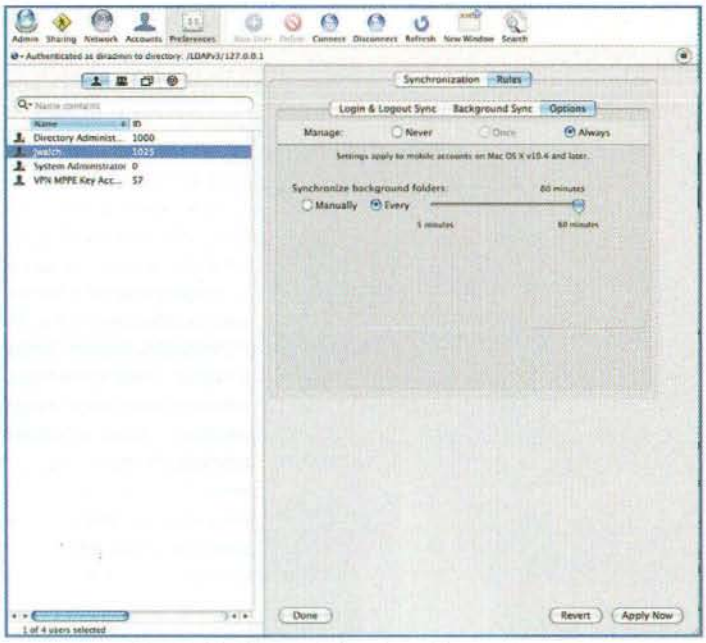
Adding new users is unchanged, and still as kludgy as ever. Apple really needs to step up its sysadmin automation capabilities here. With regard to individual user preferences, there are a few updates here. The login prefs have some new features, such as "Add network home share point" and "Merge with user's items (Mac OS X 10.4 and later). The first one is how you make sure the user's network home share point always mounts. The second one, from what I can tell, is to merge the user's personal startup items and the ones you select for them so they all start up on login. I say, "from what I can tell", because this checkbox is pretty much undocumented. Apple's PDF and discussions group don't have anything on it, (at least not as of this writing, which is about a month before you read it) and searching the Apple Knowledge Base for that item gave me no results. Spotty documentation for a user is bad. Spotty sysadmin documentation is unacceptable. If they can take the time to code

the function and the UI access for it, then Apple must take the time to document its basic function. Login Items is the only preference that you can manage for users or groups in the Workgroup Manager login preferences.

This gives me an opening to bring up a point about Workgroup Manager and the MCX (Managed Client OS X) here. There are three levels of management in MCX: User, Group, Machine. Most, but not all prefs can be set at all three levels. If you have contradictory settings, you may not get the results you want and you'll have great agony. You want to be very careful about setting the same preference on multiple levels, and do so as little as possible. Simple planning before you set will help here. Along these lines, Apple including a "Settings Check" function that would, if nothing else, highlight possible conflicts would be quite useful and very welcome.

The Media Access prefs haven't changed, but I wanted to point them out. If you have a need to limit how people can use removable media, this is an important setting. It allows you to lock down access to removable media at whatever level you require, with decent granularity. In today's SOX/GLB/HIPAA environment, being able to not allow copying of data to, or even from, removable media is an important feature, and I'm glad that Apple makes setting this up as easy as they do.

The Mobility setting is the biggest change to user management in OS X, and one that's been a long time in coming. With Mac OS X 10.3 Server, you could have "mobile" accounts, but all that did was deal with domain authentication. It did nothing for data synchronization. So, if you had a mobile account on one machine, and went to another machine, you didn't have any access to your home directory data on the first machine. Mobile accounts under Mac OS X 10.3 really only allowed you to log into the machine in situations where you were disconnected from the Open Directory network. In Mac OS X 10.4 Server, that's changed, and you now have (almost) full home directory synchronization. You can choose the items you want, or don't want to sync, and how. So, for example, you can have some directories sync on login and logout, (this would be the Windows Roaming Profile model), or have them sync in the background anywhere from once every 5 minutes to once an hour. You probably want to sync as little as possible to avoid network capacity problems. This means that if you have to switch machines a lot, you can have an almost uniform home directory setup, something that will be a major boon to mobile users, or schools, where a student may log into multiple machines throughout the day and then take a laptop home. There is one exception: Your home Library directory. That does not sync, even if you tell Workgroup Manager ***to*** sync it. This is primarily because you have some application and OS settings that use absolute hard coded paths that could break on different machines. Switching machines a lot will also make your ByHost preference management a ***lot*** more complicated. There is a workaround for this, at http://www.afp548.com/article.php?story=20050601101436323, but be warned, forcing that sync can cause problems, so be careful. However, even with that caveat, this is a feature that many have needed for a few years, and finally having it, even in this imperfect form, is a welcome change.



Portable Home Directory Sync setup

The Network preferences are a new feature, allowing administrators to set proxy preferences at the domain level, so that users can't bypass them. The Software Update prefs, also new in Mac OS X 10.4 Server, allow you to specify a local software update server you wish your users to use.

Group Accounts

At the group level, most of the account changes are wrapped around ACLs, so you can have groups within groups, etc. There are some other minor tweaks, like a group picture feature, and a comment for the group. Outside of preference features that don't exist at all in Mac OS X 10.3 Server, group preferences are unchanged.

Computer Accounts

This section has received more than a little work in Mac OS X 10.4 Server. In the ***Access*** tab, you have Mac OS X 10.4 – only options for Local – Only accounts picking workgroups from the list of groups allowed to access the machines in a computer list, and that computer administrators can disable management.

However, the specific machine information for Macs in a Computer List has grown by quite a bit. In Mac OS X 10.3 Server, about all you could do was set the computer list for a given machine, give it a name for use in Workgroup Manager, and a comment. In Mac OS X 10.4 Server, there are some new features here that work with the managed Network views. (I'll be getting to those later). So you can specify what Network view a machine can use, and the URLs that can be used to reach the computer based on what services it offers that you want used. So you can specify AFP, SMB, etc. When we go over managed Network views, you'll see how powerful this can be.

In the preferences, again, there aren't many changes that aren't Mac OS X 10.4 specific. The Login preferences are a little different for machines, in that you not only have features that only apply to the computer level, but you can now specify

login and logout scripts that run instead of, or in addition to Login/LogoutHook scripts. This is a powerful, and potentially dangerous feature, so it only works if you enable this feature for root's loginwindow defaults AND you are using trusted binding to an Open Directory domain. This makes sense, as without that precaution, you could create a rogue server that would run 'bad' scripts on Macs that you shouldn't have control over. That would be A Bad Thing. And scripts have to be 30KB in size or smaller.

All Records

This (optional) tab hasn't changed in any noticeable way for Mac OS X 10.4 Server, and while that's good from a familiarity point of view, it perpetuates one of the worst design decisions ever made, and that is the decision to hide the structure of the directory from the humans. With almost any other Directory Service, such as Active Directory, you can see the tree view of the directory, and easily manipulate the data directly. So you can drag and drop items between OUs, groups, containers, etc. Workgroup Manager still doesn't allow you to do this, which is a shame, because even with Mac OS X 10.4 Server's rudimentary OU support, being able to handle those objects easily would make directory setup and administration far easier. I get that a lot of K-12 administrators don't want or need this kind of feature, but almost every other segment does want and need it. In the Enterprise Space, Workgroup Manager is one of the weakest directory management tools on the market, and things like this are a big part of it. Apple needs to, at least for Leopard; preferable well before, let the administrators who need this ability have access to it via Apple's own tools. Otherwise, you'll never be able to really scale Open Directory past a relatively smallish size.

Managed Network Views

This is a brand new feature for Mac OS X 10.4 Server, accessed, by clicking the ***Network*** button in Workgroup Manager, and in the short version, allows you to manage what your users see when they click on the ***Network*** icon in the Mac OS X 10.4 Finder. One of the problems with the Network view in Mac OS X 10.3 is that restricting what any Mac saw was quite hard, even effectively impossible. With Mac OS X 10.4 and Mac OS X 10.4 Server, you can now manage what neighborhoods and what machines any given client in an Open Directory domain can see. If you have a small network, this is not a big issue for you. If you have a few thousand clients on multiple subnets, this is a real help in controlling spurious browsing and its associated traffic.

There are three main kinds of views:

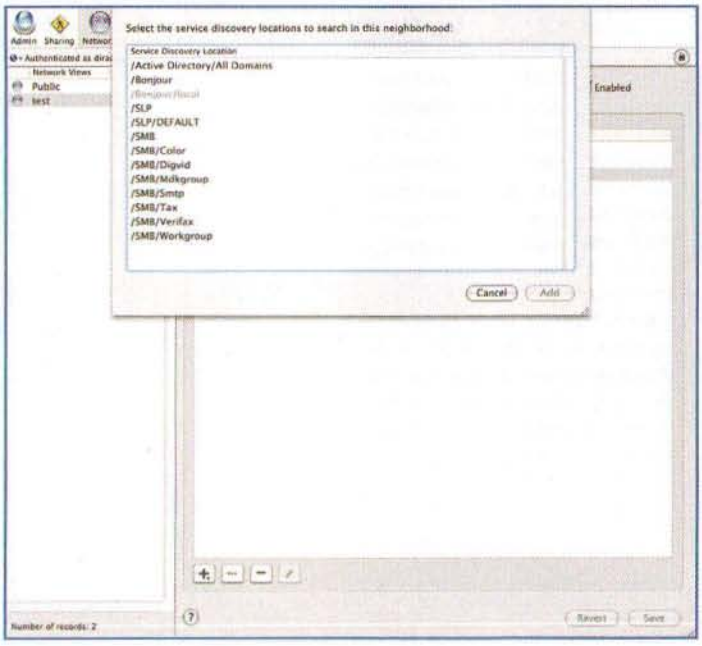
1. **Named View:** This is a network view that is visible only on those computers that you explicitly allow access for.
2. **Default View**: This is used for managed computers if there's no Named View.
3. **Public View:** This is used in lieu of the other two. If there's no Public View, but there is a Default View, that's used instead.

Within a Network View, you can have one or more of the following objects:

- **Network Neighborhood:** This is a collection, (with a name stolen right from Windows, yeah, both sides do that), which can contain any of the three object types listed here. So it can contain individual computers, other Neighborhoods, or dynamic lists. It's a catchall that you can use as a root container type.

- **Computer:** This is well, a computer. Specifically, it's a computer that Workgroup Manager knows about. You can add computers directly to a View, or to a Neighborhood. This allows you to better partition your browsing traffic, so you could, for example, have a Named View called "Directory Servers" and have only your Open Directory primaries and replicas listed there, and another Named View called "File Servers" which could contain Neighborhoods like "SMB Servers", "AFP Servers", etc.

- **Dynamic List:** This is a collection of resources that is created on the fly when you access it in the Finder. Unlike the other two, you can only create a dynamic list from existing network structures, as seen below:



Dynamic View selection

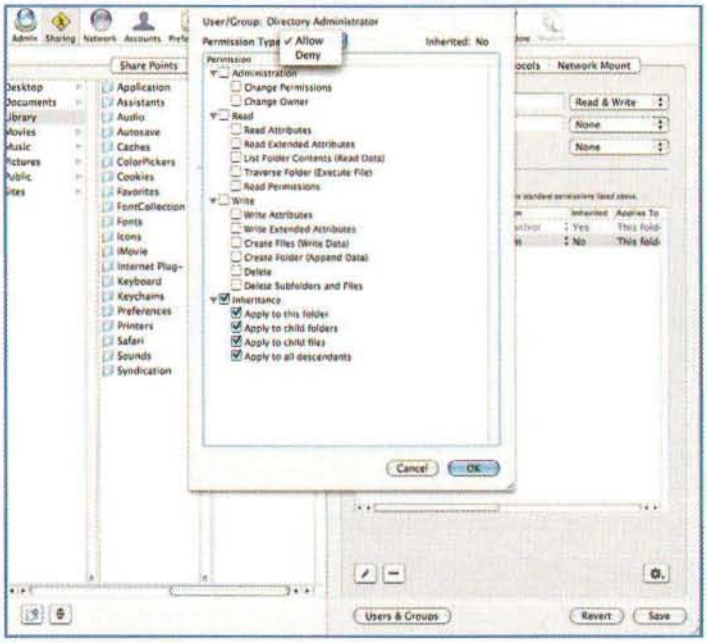
Dynamic Lists are useful when you want to limit browsing and already have service discovery structures in place. Note that you can neither manually add nor remove items from a Dynamic List. You can however, put a Dynamic List in a Neighborhood and manage access in that fashion.

Once you have created your view types and the objects they contain, you can now manage their visibility, by either having the various views add to the unmanaged Finder Network Views, or replacing the standard Finder Network

view entirely. This is a powerful tool if you need to limit access to various computers on your network, such as not allowing random access to the accounting servers, etc. It is also a good way to ensure that every computer isn't trying to browse entire network structures that they may have no need for. Limiting this kind of traffic helps enhance overall network performance, not just for the administrators, but for the users as well. If a user only needs access to four file servers, making them wait while an unmanaged browser view enumerates 150 machines is a waste of their time, not to mention bandwidth.

Sharing

ACLs are, again, the main source of changes here. In the *All* tab, you have the option to enable ACLs on a given volume. Note that this can only be done at the volume level, and outside of Workgroup Manager, you can only do this via fsaclctl. Once that's done, you can then apply ACLs to specific folders and files within that volume at your discretion. This can be done outside of any sharing you may implement for a folder, and unlike sharing you can set ACLs on files too. Remember that I've been saying ***be careful with ACLs*** a lot? This is why:



Workgroup Manager ACL settings dialog

ACLs give you a lot of capabilities, but setting them willy-nilly, and not watching how you set groups within groups, or tracking who is in what group for which ACL entry will, not can, but *will* cause you problems. One benefit of Apple's implementation is that if you do get into trouble you can, as a last resort, turn ACLs *off* and start over again. I know Windows Admins who would love to do that, because they accidently created an ACL for a folder that not only keeps everyone out, but won't even let them delete it without reformatting the drive or other very drastic action.

I don't want to scare you away from ACLs, but you need to give them a lot of respect. As the disclaimer says, ***not intended for amateurs.***

Outside of ACLs, sharing hasn't changed much. There's some improvement to the strict locking for SMB shares, (Note: You should only enable oplocks on shares that will only be touched by Windows clients.), but from the Workgroup Manager point of view, sharing's pretty much the same as it was in Mac OS X 10.3 Server, you just have juicy ACL goodness.

Conclusion

That's it for part two of this series. It's quite a bit shorter than the Server Admin part was, but then Server Admin is the basis for most of what Workgroup Manager does, so there's not as much change to talk about. The final part of this series will show up next month, and cover, well, everything else.

Bibliography and References

As with the first article in this series, almost everything in this article can be found in Apple's Server Documentation, at http://www.apple.com/server/documentation/. What little isn't there, I pried from the ridiculously busy brains of people like Schoun Regan of I.T. Instruction, Michael Bartosh of 4am Media, and Joel Rennich of AFP548.com. Schoun and Michael are the authors of the two best books on Mac OS X Server available, the Visual Quickstart Guide to Mac OS X Server, and Essential Mac OS X Server Administration, respectively. Buy them both, they're great books. If you read any of my columns and ***don't*** regularly read AFP548.com, then you're missing out on a fantastic resource. All three of these guys, Schoun, Michael, and Joel are Apple Trainers too, a great reason to take the Apple courses if you haven't yet. Those courses are taught by the best folks in the Mac market, and well worth their cost.

About The Author

John Welch <jwelch@bynkii.com> is an IT Staff Member for Kansas City Life Insurance, a Technical Strategist for Provar, (http://www.provar.com/) and the Chief Know-It-All for TackyShirt, (http://www.tackyshirt.com/. He has over fifteen years of experience at making Macs work with other computer systems. John specializes in figuring out ways in which to make the Mac do what nobody thinks it can, showing that the Mac is a superior administrative platform, and teaching others how to use it in interesting, if sometimes frightening ways. He also does things that don't involve computertry on occasion, or at least that's the rumor.

Exclusive File Access In Mac OS X

By Paul T. Ammann

Opening a file from classic Mac OS (pre Mac OS X) with fsWrPerm, fsRdWrPerm, or the default fsCurPerm, meant that any other application trying to open that same file with write access would not be able to do so. Usually, an fsRdWrPerm error would be returned when other attempts were made to open the file for write access, though attempts to open such a file for read-only access would succeed. This default behavior allows for one "write" and multiple "readers" of the file.

Mac OS X's BSD subsystem does not enforce file read/write privileges in the same way as classic Mac OS. Opening a file for writing does not ensure other processes can not write to the same file. The default behavior of BSD allows for multiple "writers" to a single file. As a result, opening a file via PBHOpenDF, PBHOpenRF, PBHOpen, PBOpenFork, FSOpenFork, HOpen, etc., on a local volume and passing in a permissions value of fsCurPerm, fsWrPerm, or fsRdWrPerm does not guarantee exclusive file access on Mac OS X. On Mac OS X, subsequent Open calls to open a file with write permission may succeed without error. Similarly, the PBLockRange() routines may not actually guarantee byte ranges that cannot be modified by other processes. Because these routines may return without error, you should check out the availability of exclusive file access (see "Checking Availability of Exclusive File Access") before making any assumptions about the underlying file access. If the "supports advisory locks" feature is not available, your application will not know if the file is already in use by another application.

AppleShare servers and Personal File Sharing on Mac OS X do enforce exclusive file access and range locking for volumes accessed over the network. However, this functionality is only available when accessing files over a networked file sharing connection and is not available to applications running on the server itself.

Guidelines for Working with Non-exclusivity

You should realize that many applications relied on the behavior of the classic Mac OS File Manager to prevent multiple applications from writing to the same file (or to control write access through byte range locking). Since that behavior is not implemented in all versions of Mac OS X, some common workarounds that you may wish to use in your code are described below. BSD was designed without exclusive locks in order to prevent denial of service attacks in which one process opens a file with an exclusive lock which may be required by another process, effectively blocking the other process.

Checking Availability of Exclusive File Access

Mac OS X will enforce exclusive file access, i.e. one writer and many readers of a file, through its application frameworks, Carbon, Cocoa, and Java by enforcing BSD advisory locks as though they are exclusive. The "supports advisory locks" feature is defined if both the OS and file system for the volume in question support advisory locks. In this case, the default behavior of the application frameworks is to open files with exclusive access when opened as writable. Applications built on these frameworks automatically get this functionality and do not need to be modified. When the conditions are met to support exclusive file access, PBLockRange will also call down through to the BSD advisory locks. Since PBLockRange will be based on BSD advisory locks at this point, range locks can be applied to local files as well as those on file servers.

Since not all versions of Carbon on Mac OS X support exclusive file access nor do all file systems support BSD advisory locks, you should check a couple of things before making assumptions about the underlying file access behavior. You should only assume these features are available if the gestalt bit, gestaltFSSupportsExclusiveLocks, as well as the GetVolParms bit, bSupportsExclusiveLocks, are both set. For instance, the Carbon Framework File Manager routines support advisory locks by default when SupportsExclusiveFileAccess returns true.

```
#ifndef gestaltFSSupportsExclusiveLocks
   #define    gestaltFSSupportsExclusiveLocks    15
   #define    bSupportsExclusiveLocks            18
#endif

Boolean    SupportsExclusiveFileAccess( short vRefNum )
{
   OSErr                    err;
   GetVolParmsInfoBuffer    volParmsBuffer;
   HParamBlockRec           hPB;
   long                     response;
   Boolean                  exclusiveAccess    =
false;

   err = Gestalt( gestaltSystemVersion, &response );
   if ( (err == noErr) && (response < 0x01000) )
   {
      err = Gestalt( gestaltMacOSCompatibilityBoxAttr,
&response );
      if ( (err != noErr)
      || ((response & (1 <<
gestaltMacOSCompatibilityBoxPresent)) == 0) )
         return( true );        //    Running on Mac
OS 9, not in Classic
   }

   err = Gestalt( gestaltFSAttr, &response );
   if ( (err == noErr)
      && (response & (1L <<
gestaltFSSupportsExclusiveLocks)) )
   {
      hPB.ioParam.ioVRefNum      = vRefNum;
      hPB.ioParam.ioNamePtr      = NULL;
      hPB.ioParam.ioBuffer       = (Ptr)
&volParmsBuffer;
      hPB.ioParam.ioReqCount     = sizeof( volParmsBuffer
);
      err = PBHGetVolParmsSync( &hPB );
      if ( err == noErr )
         exclusiveAccess =
               (volParmsBuffer.vMExtendedAttributes
               & (1L << bSupportsExclusiveLocks)) !=
0;
   }

   return( exclusiveAccess );
}
```

To check if a volume supports byte range locking via PBLockRange you should check the bHasOpenDeny bit returned from GetVolParms. See Technical Note FL37 (http://developer.apple.com/technotes/fl/fl_37.html) for more information about PBLockRange details.

Common Workarounds

The following two techniques are frequently used to work around this issue on platforms that do not enforce exclusive file access.

Lockfiles

A common approach used by many developers is to create a "lockfile" in the same directory as the file being opened. Whenever opening a file, "foo", for instance, with write access you first try to create a lockfile, "foo.lock", in the same location. If the file creation fails because the file already exists, you assume "foo" is already open by another application. Upon closing "foo" the application is also responsible for deleting "foo.lock". A strength of this technique is it only makes one assumption about the underlying file system: the file creation operation is atomic. The obvious weakness is that since there is no OS support for this method, each application is responsible for implementing its own lock file mechanism, and there are no agreed upon standards or conventions for the naming of lock files.

Edit a Copy

Another workaround relies on operating on a unique copy of the file. When a file is opened for editing, a duplicate of the file is created in /tmp directory with a unique name, and opened. When the user tries to save the document, the modification date of the original is matched against the date cached during the open of the file. If it has changed, you know the file was modified.

MAC OS X Solutions

BSD Advisory Locking

Although Mac OS X's BSD subsystem does not implement provisions for exclusive write access, (i.e. mandatory locks), it does provide advisory locks. An advisory lock is a voluntary locking mechanism in which the underlying file system maintains a linked list of record locks. As long as your application and other applications respect the locks, only one application at a time will have write access to a particular file. Since these locks are voluntary it is the choice/responsibility of the application developer to respect or ignore advisory locks. If you would like to use advisory locks, this can be done by

following the instructions later in this article. By accessing files through the application frameworks (Carbon, Cocoa, Java), in versions of the OS supporting the advisory locks feature in frameworks, this will be provided automatically if you use the framework's file access methods.

Applications that call BSD file I/O functions directly will not gain this behavior for free, and therefore should be revised to set and respect advisory locks by specifying the appropriate flags when opening a file.

You should evaluate changing calls from:

```
fd = open( "./foo", O_RDWR );
```

to

```
fd = open( "./foo", O_RDWR + O_EXLOCK + O_NONBLOCK );
```

Where, O_EXLOCK means Atomically obtain an exclusive lock, and O_NONBLOCK means Do not block on open or for data to become available or Do not wait for the device or file to be ready or available.

Implementing Advisory Locking

Anywhere you are calling the System.framework version of open(2) with write access, you should modify the parameters to include the "O_EXLOCK + O_NONBLOCK" flags, and handle errors being returned, where they may have succeeded in the past. The open(2) call will then fail if the file has already been opened for exclusive access by another process.

Advisory locks are associated with a process and a file. This has two implications:

- When a process terminates all its locks are released.
- Whenever a descriptor is closed, any locks on the file referenced by that descriptor are released.

Implementing Byte Range Locking

BSD also provides advisory byte range locking support through the fcntl() function. By using advisory locking, your applications will be able to work in a cooperative with Carbon, Classic, and other applications in the future. In these circumstances, files should be opened with the O_EXLOCK set and then ranges locked through the fcntl() call.

Stevens' "Advanced Programming in the UNIX Environment" (page 367) describes some techniques for using the UNIX service fnctl() to lock portion of a file for reading and writing (Stevens, 1999, p. 367).

Warning: A file lock request which is blocked can be interrupted by a signal. In this case the lock operation returns EINTR. Thus you may think you got a lock when you really did not. A solution is to block signals when locking. Another solution is to test the value returned by the lock operation and relock if the value is EINTR. Another solution, which we adopt here, is to do nothing about it.

Recording Locking is the term normally used to describe the ability of a process to prevent other processes from modifying a region of a file while the process is reading or modifying that portion of the file. BSD provides access to its record locking mechanism through the fcntl function:

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

/*
 * Returns:
 *     -1 on error
 */
int fcntl(int filedes, int cmd, ... /* struct flock
*flockptr */ );
```

We'll start with the third argument (flockptr), which points to a flock structure:

```
struct flock {
    short l_type;
/* F_RDLCK (shared read lock), or
 * F_WRLCK (shared write lock), or
 * F_UNLCK (unlocking a region)
 */

off_t l_start;
/* offset in bytes, relative to l_whence */

short l_whence;
/* SEEK_SET: file's offset is set to
 *    l_start bytes from beginning of file
 * SEEK_CUR: file's offset is set to its current
 *    value plus the l_start (which can
 *    be + or -)
 * SEEK_END: file's offset is set to the size of
 *    the file plus the l_start (which can
 *    be + or -)
 */

off_t l_len;
/* length of region, in bytes
 * special case: if (l_len == 0), it means that
 * the lock extends to the largest possible
 * offset of the file. This allows us to lock a
 * region starting anywhere in the file, up
 * through and including any data that is
 * appended to the file
 */

pid_t l_pid;
/* returned when cmd = F_GETLK */
}
```

This structure describes:

- The type of lock desired (i.e. read lock, write lock, unlock)
- The starting byte offset of the region being locked or unlocked (l_start and l_whence)
- The size of the region (l_len)

To lock an entire file, set l_start and l_whence to point to the beginning of the file (i.e. l_start= 0, l_whence= SEEK_SET), and specify a length (l_len) of 0.

Any number of processes can have a shared read lock on a given byte, but only one process can have an exclusive write lock on a given byte. To obtain a read lock the descriptor must be open for reading, and the region cannot have an exclusive write lock. To obtain a write lock the descriptor must be open for writing, and the region cannot have an exclusive write lock nor any read locks.

Now, we will describe the second parameter (cmd) for fcntl. The possible commands and what they mean are described in the following:

- F_GETLK: Determine if the lock described by flockptr is blocked by some other lock. If a lock exists that would prevent ours from being created, the information on that existing lock overwrites the information pointed to by flockptr. If no lock exists that would prevent ours from being created, the structure pointed to by flockptr is left unchanged except for the l_type member, which is set to F_UNLCK.

- F_SETLK: Set the lock described by flockptr. If we are unable to obtain a lock (because of previous locks already granted for the region) then fcntl returns -1 and errno is set to either EACCES or EAGAIN.

- F_SETLKW: This command is a blocking version of F_SETLK (the W in the command means "wait"). If the requested read lock or write lock cannot be granted because another process currently has some part of the requested region locked, the calling process is put to sleep. This sleep is interrupted if a signal is caught.

Be aware that testing for a lock with F_GETLK and then trying to obtain that lock with F_SETLK or F_SETLKW is not an atomic operation. We have no guarantee that between the two fcntrl calls some other process won't come in and obtain the same lock.

To save ourselves the trouble of allocating a flock structure and filling in all the elements each time, Stevens defines the function lock_reg and a number of macros that call it. Notice that the macros shorten the number of parameters by two, and save us from having to remember the F_* constants mentioned above.

```
#define read_lock(fd, offset, whence, len)      \
        lock_reg  (fd, F_SETLK,  F_RDLCK, offset,
whence, len)

#define readw_lock(fd, offset, whence, len)     \
        lock_reg  (fd, F_SETLKW, F_RDLCK, offset,
whence, len)

#define write_lock(fd, offset, whence, len)     \
        lock_reg  (fd, F_SETLK,  F_WRLCK, offset,
whence, len)

#define writew_lock(fd, offset, whence, len)    \
        lock_reg  (fd, F_SETLKW, F_WRLCK, offset,
whence, len)

#define un_lock(fd, offset, whence, len)        \
        lock_reg  (fd, F_SETLK,  F_UNLCK, offset,
whence, len)


pid_t    lock_test(int, int , off_t , int , off_t );

#define    is_readlock(fd, offset, whence, len) \
             lock_test(fd, F_RDLCK, offset, whence, len)
#define    is_writelock(fd, offset, whence, len) \
             lock_test(fd, F_WRLCK, offset, whence, len)
```

```
int lock_reg(int fd, int cmd, int type, off_t offset,
int whence, off_t len)
{
    struct flock lock;

    lock.l_type    = type;      /* F_RDLCK, F_WRLCK,
F_UNLCK        */
    lock.l_start   = offset;    /* byte offset, relative
to l_whence */
    lock.l_whence = whence;     /* SEEK_SET, SEEK_CUR,
SEEK_END       */
    lock.l_len     = len;       /* #bytes (0 means to
EOF)           */

    return ( fcntl(fd, cmd, &lock) );
}


pid_t    lock_test(int fd, int type, off_t offset, int
whence, off_t len)
{
  struct flock lock;
  lock.l_type = type;      /* F_RDLCK or F_WRLCK */
  lock.l_start = offset;   /* byte offset relative to
l_whence */
  lock.l_whence = whence; /* SEEK_SET, SEEK_CUR,
SEEK_END */
  lock.l_len = len;        /* #bytes (0 means to EOF) */
  if (fcntl(fd,F_GETLK,&lock) < 0){
    perror("fcntl"); exit(1);}
  if (lock.l_type == F_UNLCK)
    return (0);          /* false, region is not locked
by another process */
  return (lock.l_pid); /* true, return pid of lock owner
*/
}
```

There are three important rules regarding automatic inheritance and release of record locks:

- Locks are associated with a process and a file. When a process terminates, all its locks are released. Whenever a descriptor is closed, any locks on the file referenced by that descriptor for that process are released.

- Locks are never inherited by the child across a fork (otherwise we could end up with two processes sharing a write lock)

- Locks may be inherited by a new program across an exec. This is not required by BSD and is therefore machine dependent

References

Stevens, Richard W. (1999). Advanced Programming in the UNIX Environment

Massachusetts: Addison Wesley Longman, Inc.

ISBN: 0201563177

About The Author

Paul Ammann has been working in IT for almost 20 years. He is happily married to his wife Eve for 6 years. He finds writing the author's bio the toughest part the article.

THREADS

PERFORMING QTKIT OPERATIONS ON THREADS

In the previous three articles (in MacTech, May, June, and July 2005), we have investigated the QTKit, Apple's new framework for accessing QuickTime capabilities in Cocoa applications and tools. We have used this framework — introduced in Tiger and also available in Panther with QuickTime 7 — primarily to build a sample application that can open and display one or more QuickTime movie files (or other media files openable as QuickTime movies) and that supports the standard suite of movie editing and document management operations.

Introduction

The sample application that we have been developing, KitEez, performs really quite well. Files open and display quickly, and cut-and-paste editing works as smoothly as could be expected. And, as mentioned in one of those previous articles, QTKit-based drag-and-drop performance in most instances greatly exceeds that provided by the standard movie controller component.

But, truth be told, we haven't really asked KitEez to handle any potentially slow operations. Currently it can open only files selectable in the file-opening dialog box, which means that we're not likely to see any real delay when opening a movie file. If KitEez were to support files specified by remote URLs, we'd definitely see some slowdown in movie opening. Also, some operations that we can perform using QuickTime functions — like importing or exporting movies and large pictures — can take a considerable amount of time. If KitEez were to support those sorts of operations, we'd want to make sure that the user was able to keep working while they are churning away. That generally means that we'd need to learn how to perform QTKit operations on a background thread.

In this article, we're going to take a look at threading and QTKit. Since QTKit is built on top of QuickTime, it inherits whatever limitations exist in QuickTime with regard to being able to perform operations on background threads. QuickTime has supported threaded operations since Mac OS X 10.3 and later, with QuickTime 6.4 and later. That is to say, it's now possible to move potentially time-consuming operations to a background thread, thereby freeing up the main thread for user interaction. Gone are the days when importing or exporting a large file invariably ties up your QuickTime application.

To achieve this, we'll need to use a few new QuickTime functions to set up the QuickTime environment on a background thread. We'll need to learn how to safely move QTKit objects between threads, and we'll need to be careful about which operations are performed on a background thread. But with these three issues covered, we should be able to provide a far better user experience than is possible in a single-threaded application.

Before we begin, a couple of provisos. It would be

good if you were already familiar with application threading in general and with Cocoa's NSThread class in particular. If not, don't worry; you should be able to pick up what you need along the way. Just don't expect a detailed discussion of threading models here. Also, I will be focusing on invoking QTKit and other Cocoa methods on background threads; see the documents mentioned in the References section for more complete information about threading in QuickTime generally.

Opening Movies Revisited

As mentioned above, our sample application KitEez currently can open only files selectable in the file-opening dialog box, which generally means that it will be opening only local files. In the previous article, we saw that we can also use the initWithURL:error: method to open a remote file specified by a URL. To elicit a URL from the user, we might display a dialog box like the one in Figure 1. (Adding this dialog box to KitEez is left as an exercise for the reader.)

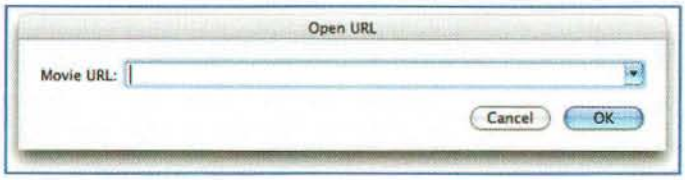


Figure 1: A URL dialog box

As you know, initWithURL:error: operates asynchronously, just like all the QTKit movie-opening methods; that is to say, it returns almost immediately, so that our application can continue processing while the movie data loads. So we might be tempted to immediately assign the QTMovie object to the movie view in our document window, like this:

```
if ([QTMovie canInitWithURL:url]) {
  movie = [[QTMovie alloc] initWithURL:url error:nil];
  if (movie)
    [movieView setMovie:movie];
}
```

This would work fine, from the standpoint of having QuickTime download the movie data without further intervention from our application. Experience has shown, however, that it's best to defer the setMovie: call until a sufficient amount of movie data has been downloaded. To do that, we can install a notification handler for the QTMovieLoadStateDidChangeNotification notification, like this:

```
if ([QTMovie canInitWithURL:url]) {
  movie = [[QTMovie alloc] initWithURL:url error:nil];
  if (movie)
    [[NSNotificationCenter defaultCenter] addObserver:self
                selector:@selector(loadStateChanged:)
                name:QTMovieLoadStateDidChangeNotification
                object:movie];
}
```

Whenever the *load state* of the downloading movie changes, the loadStateChanged: method will be called. (See "Loaded" in *MacTech*, September 2002 for a complete discussion of movie load states.) One easy implementation of loadStateChanged: is shown in Listing 1.

Listing 1: Handling load state-changed notifications

```
- (void)loadStateChanged:(NSNotification *)notification
{
  if ([[movie attributeForKey:QTMovieLoadStateAttribute]
                longValue] >= kMovieLoadStatePlayable) {
    [[NSNotificationCenter defaultCenter]
                removeObserver:self
                name:QTMovieLoadStateDidChangeNotification
                object:movie];

    [movieView setMovie:movie];
    [movie release];

    [[movieView movie] play];
  }
}
```

As you can see, we wait until the load state of the movie reaches the kMovieLoadStatePlayable level, at which time we remove the notification listener for the specified notification, call setMovie:, release our QTMovie object (since the movie view will retain it), and then start the movie playing. The kMovieLoadStatePlayable constant is defined in the QuickTime header file Movies.h. Here is the complete set of load state constants:

```
enum {
  kMovieLoadStateError                = -1L,
  kMovieLoadStateLoading              = 1000,
  kMovieLoadStateLoaded               = 2000,
  kMovieLoadStatePlayable             = 10000,
  kMovieLoadStatePlaythroughOK        = 20000,
  kMovieLoadStateComplete             = 100000L
};
```

We need to defer the call to setMovie: until the movie is playable to work around a bug in the first release of QTKit. If we didn't do this, and instead just called setMovie: immediately after the call to initWithURL:error:, we would find that the movie view would not automatically redraw itself once any video data had arrived. The workaround is simple enough and indeed provides an easy way for us to start the movie playing at an appropriate time.

It's also useful to know how to get the load state of a QTMovie object when we want to add importing and exporting support to our applications. The reason is simple: whenever we want to export or flatten or save a movie, we need to have all of its movie and media data

on hand. The QTMovie method that we use to export or flatten a movie, writeToFile:withAttributes:, internally checks the movie's load state and returns NO if it's not at least kMovieLoadStateComplete. But we might need to make this check ourselves, when adjusting some menu items. If not all the movie and media data is available, then for instance the Export... menu item should not be enabled. Listing 2 shows a segment of a document class' override of the validateMenuItem: method.

Listing 2: Enabling menu items

```
- (BOOL)validateMenuItem:(NSMenuItem *)menuItem
{
  BOOL valid = NO;
  SEL action;

  action = [menuItem action];

  if (action == @selector(doExport:))
    valid = ([[movieView movie]
      attributeForKey:QTMovieLoadStateAttribute] longValue]
      >= kMovieLoadStateComplete);

  // other lines omitted...

  return valid;
}
```

Keep in mind that we have not yet reached a position where we need to move any processing out of the main thread and into a secondary or background thread. The periodic movie tasking that is required to keep a remote movie steadily downloading happens automatically on the main thread and does not generally consume so much processor time that the responsiveness of the main thread is adversely impacted. So, although opening a movie specified by a URL may take a significant amount of time, QuickTime (and hence QTKit) already knows how to do that asynchronously without blocking the main thread.

Importing and Exporting

When we move into the realm of movie importing and exporting — that is, converting potentially large amounts of data — we cross an important threshold. Although it is certainly possible to export a movie by calling writeToFile:withAttributes: on the main thread, it isn't really advisable, since the call would execute synchronously. Listing 3 shows how *not* to define the doExport: method referenced in Listing 2.

Listing 3: Exporting a movie as 3GPP

```
- (IBAction)doExport:(id)sender
{
  NSDictionary *dict = [NSDictionary
      dictionaryWithObjectsAndKeys:
      [NSNumber numberWithBool:YES], QTMovieExport,
      [NSNumber numberWithLong:kQTFileType3GPP],
      QTMovieExportType, nil];

  [[movieView movie] writeToFile:@"/tmp/sample.3gp"
      withAttributes:dict];
}
```

If the movie is very large, this method could take quite a while to complete. During that time, the user would be unable to do anything with our application except move windows around. Not very exciting.

A slightly better solution involves using the movie:shouldContinueOperation:withPhase:atPercent:withAttributes: delegate method described briefly in the previous article. As I mentioned, this is a wrapper around QuickTime's movie progress function, which we have used in earlier articles to display a dialog box showing the progress of the export and to allow the user to cancel the operation. Figure 2 shows the sheet we'll display from within that delegate method.

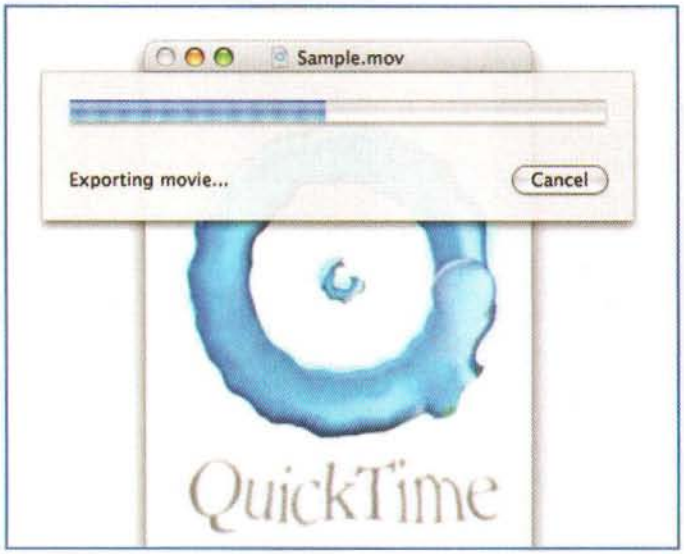


Figure 2: A progress sheet

We could implement this delegate method as shown in Listing 4.

Listing 4: Displaying a cancelable progress sheet

```
- (BOOL)movie:(QTMovie *)movie
    shouldContinueOperation:(NSString *)op
    withPhase:(QTMovieOperationPhase)phase
    atPercent:(NSNumber *)percent
    withAttributes:(NSDictionary *)attributes
{
  OSErr err = noErr;
  NSEvent *event;
  double percentDone = [percent doubleValue] * 100.0;

  switch (phase) {
    case QTMovieOperationBeginPhase:
      // set up the progress panel
      [progressText setStringValue:op];
      [progressBar setDoubleValue:0];

      // show the progress sheet
      [NSApp beginSheet:progressPanel
        modalForWindow:[movieView window] modalDelegate:nil
        didEndSelector:nil contextInfo:nil];
      break;
    case QTMovieOperationUpdatePercentPhase:
      // update the percent done
      [progressBar setDoubleValue:percentDone];
      [progressBar display];
```

```
            break;
        case QTMovieOperationEndPhase:
            [NSApp endSheet:progressPanel];
            [progressPanel close];
            break;
    }

    // cancel (if requested)
    event = [progressPanel
        nextEventMatchingMask:NSLeftMouseUpMask
        untilDate:[NSDate distantPast]
        inMode:NSDefaultRunLoopMode dequeue:YES];
    if (event && NSPointInRect([event locationInWindow],
                               [cancelButton frame])) {
        [cancelButton performClick:self];
        err = userCanceledErr;
    }

    return (err == noErr);
}
```

This is certainly a better solution than having no sheet at all, but it's really not satisfactory. Just distracting the user with a progress bar is not going to make the export go any faster or be any less synchronous. And the manner in which we check for clicks on the Cancel button is not really very good, even if it ***is*** the best we can hope for in a non-threaded application.

Threaded Exporting

So we really do need to move to a multithreaded application if we want to be able to provide acceptable responsiveness in our application's user interface while performing potentially lengthy operations like exporting a movie. As we'll see, spawning a thread to execute some code in a Cocoa application is as easy as calling the NSThread method detachNewThreadSelector:toTarget:withObject:. The complexities we shall encounter arise from the fact that QuickTime was not originally written to be thread-safe, and making it work in a threaded environment requires some assistance from the application developer. For some of the theory, consult the documents listed at the end of this article, particularly the Tech Note on threading QuickTime applications. For the moment, we will content ourselves with the practical implications of that theory. In summary, they are these:

(1) Before any QuickTime APIs (including QTKit methods) can be called on a background thread, the function EnterMoviesOnThread must be called on that thread.
(2) After all QuickTime APIs (including QTKit methods) have been called on a background thread, the function ExitMoviesOnThread must be called on that thread.
(3) A movie created on one thread that is to be accessed on some other thread must first be detached from the first thread (by calling DetachMovieFromCurrentThread) and attached to that other thread (by calling AttachMovieToCurrentThread).
(4) QuickTime APIs (including QTKit methods) executing on a background thread may internally attempt to instantiate components that are not thread-safe; when that happens, the result code componentNotThreadSafeErr (-2098) will be returned. In that case, you might want to retry the operation on the main thread.

Implication (4) has an important corollary. Recall from the first article on QTKit that a QTMovie object is a Cocoa representation of a QuickTime movie ***and*** a QuickTime movie controller. That is to say, a QTMovie object is associated with a Movie instance and a MovieController instance. Currently, no movie controller components are thread-safe. This means:

(5) All QTMovie objects must be created on the main thread.

In theory, creating a QTMovie object on the main thread and then migrating it to a background thread is no less dangerous than creating it on a background thread. Experience has shown, however, that it appears to be safe to call QTMovie methods on a QTMovie object that has been thus migrated. At any rate, this is the best we can do given the current state of the QTKit and the underlying movie controller components.

Transferring Movies Between Threads

To see how these implications play out in practice, let's walk through some code that exports a QuickTime movie on a background thread. If the application's Export... menu item is connected to the doExport: method, we can start the export process by calling detachNewThreadSelector:toTarget:withObject:, passing the selector of the application's doExportOnThread: method. The doExport: method is shown in Listing 5.

Listing 5: Starting a background export

```
- (IBAction)doExport:(id)sender
{
    NSSavePanel *savePanel = [NSSavePanel savePanel];
    Movie qtMovie = [[movieView movie] quickTimeMovie];
    int result;
    OSErr err = noErr;

    result = [savePanel runModal];
    if (result == NSOKButton) {
        SEL sel = @selector(doExportOnThread:);

        [[movieView movie] stop];
        err = DetachMovieFromCurrentThread(qtMovie);
        if (err)
            return;

        // show the progress sheet
        [NSApp beginSheet:progressPanel
            modalForWindow:[movieView window] modalDelegate:nil
            didEndSelector:nil contextInfo:nil];

        [NSThread detachNewThreadSelector:sel toTarget:self
                withObject:[savePanel filename]];
    }
}
```

There is nothing particularly noteworthy here except for the call to DetachMovieFromCurrentThread. The doExport: method is called on the main thread, so we need to explicitly detach the Movie associated with the QTMovie object from the main thread so that it can later be attached to a background thread. Notice also that we have moved the call to display the progress sheet out of the delegate method and into the doExport: method.

Accessing Movies on Background Threads

Now let's start building the doExportOnThread: method. Its declaration should look like this:

```
- (IBAction)doExportOnThread:(id)sender;
```

Here, the sender object is in fact the name of the file into which the exported movie is to be written (as you can see from Listing 5). Since doExportOnThread: is to be run on a background thread, it needs to create and release an autorelease pool, and it needs to call EnterMoviesOnThread and ExitMoviesOnThread. Listing 6 shows the basic skeleton of a method that is to support QuickTime calls on a background thread.

Listing 6: Exporting a movie in the background (skeleton version)

```
- (IBAction)doExportOnThread:(id)sender
{
  NSAutoreleasePool* pool = [[NSAutoreleasePool alloc]
                                        init];

  OSErr err = EnterMoviesOnThread(0);
  if (err)
    goto bail;

  err = AttachMovieToCurrentThread([movie quickTimeMovie]);
  if (err)
    goto bail;

  // QuickTime/QTKit calls can go in here....

  DetachMovieFromCurrentThread([movie quickTimeMovie]);
  ExitMoviesOnThread();

bail:
  [pool release];
  [NSThread exit];
}
```

The EnterMoviesOnThread function is declared like this:

```
OSErr EnterMoviesOnThread (UInt32 inFlags);
```

Currently only one bit in the inFlags parameter is defined, namely kQTEnterMoviesFlagDontSetComponentsThreadMode. Setting this flag forces no change to be made to the Component Manager threading mode. By default, EnterMoviesOnThread automatically sets the Component Manager threading mode to kCSAcceptThreadSafeComponentsOnlyMode, which indicates that only thread-safe components shall be allowed. Since this is the mode we desire, we'll pass 0 when we call EnterMoviesOnThread.

When we are finished using QuickTime API calls on a particular background thread, we need to call ExitMoviesOnThread, which takes no parameters. Each call to EnterMoviesOnThread must be balanced by a call to ExitMoviesOnThread.

The main thing we need to do is add some code that exports the specified movie into the filename indicated by the sender parameter. That code might look like this:

```
NSDictionary *dict = [NSDictionary
  dictionaryWithObjectsAndKeys:
    [NSNumber numberWithInt:1], QTMovieExport,
    [NSNumber numberWithInt:kQTFileType3GPP],
                                  QTMovieExportType, nil];
[movie writeToFile:sender withAttributes:dict];
```

Once the writeToFile:withAttributes: method completes, we need to make sure that the progress panel is removed and that the movie is transferred back to the main thread. We can do that by adding one more line of code, just before the bail label:

```
[self performSelectorOnMainThread:
    @selector(finishedExporting) withObject:nil
    waitUntilDone:YES];
```

Listing 7 shows our implementation of the finishedExporting method.

Listing 7: Cleaning up after an export operation

```
-(void)finishedExporting
{
  [NSApp endSheet:progressPanel];
  [progressPanel close];

  AttachMovieToCurrentThread([movie quickTimeMovie]);
}
```

And so we are done building the doExportOnThread: method and the methods it calls.

Handling the Cancel Button

One final task awaits us, namely displaying the progress sheet, updating its progress bar, and handling clicks on the Cancel button. It turns out that we already have most of the code we need at hand, in the form of our movie:shouldContinueOperation:withPhase:atPercent:withAttributes: delegate method. The first thing we need to change is the cheesy way in which we detect clicks on the Cancel button. In the nib file, we configure that button to initiate the doCancel: action, implemented in Listing 8.

Listing 8: Handling clicks on the Cancel button

```
- (IBAction)doCancel:(id)sender
{
  cancel = YES;
}
```

Then, in the delegate method, we toss all the code that looks for mouse-up events in the button and replace it with this easy test:

```
if (cancel)
  err = userCanceledErr;
```

Also, we cannot set the values of the text field and the progress bar from within the delegate method, because this delegate method wraps a movie progress function, which is called on the same thread as the export operation — that is, on a background thread. In general, a background thread must never directly alter the application's user interface. What we need to do is have the delegate method use the NSObject method performSelectorOnMainThread:withObject:waitUntilDone:, as we did earlier. So we'll rework the various case blocks like this:

```
case QTMovieOperationUpdatePercentPhase:
  // update the percent done
  [self updateProgress:op toNumber:percent];
  break;
```

Listing 9 shows our definition of updateProgress:toNumber:.

Listing 9: Sending UI updates to the main thread

```
- (void)updateProgress:(NSString*)msg
              toNumber:(NSNumber*)value
{
  NSDictionary* dict = [NSDictionary
      dictionaryWithObjectsAndKeys:msg, @"msg",
                                   value, @"value", nil];
  [self performSelectorOnMainThread:
        @selector(updateProgressInMainThread:)
        withObject:dict waitUntilDone:NO];
  return ;
}
```

Finally, Listing 10 shows the code we run on the main thread to update the items in the progress panel.

Listing 10: Updating the progress panel

```
- (void)updateProgressInMainThread:(NSDictionary*)dict
{
  NSString* msg = [dict objectForKey:@"msg"];
  double value = [[dict objectForKey:@"value"] doubleValue]
                                        * 100.0;

  [progressText setStringValue:msg];
  [progressBar setDoubleValue:value];
}
```

For completeness, let's take a last look at the revised version of the delegate method we are using to drive the progress updating (Listing 11).

Listing 11: Displaying a cancelable progress sheet (revised)

```
- (BOOL)movie:(QTMovie *)movie
    shouldContinueOperation:(NSString *)op
    withPhase:(QTMovieOperationPhase)phase
    atPercent:(NSNumber *)percent
    withAttributes:(NSDictionary *)attributes
{
  OSErr    err = noErr;

  switch (phase) {
    case QTMovieOperationBeginPhase:
      // set up the progress panel
      [self updateProgress:op toNumber:
                          [NSNumber numberWithLong:0]];
      break;
    case QTMovieOperationUpdatePercentPhase:
      // update the percent done
      [self updateProgress:op toNumber:percent];
      break;
    case QTMovieOperationEndPhase:
      break;
    }

  if (cancel)
    err = userCanceledErr;

  return (err == noErr);
}
```

Conclusion

In this article, we've taken a look at executing QTKit methods on secondary threads, in an effort to offload lengthy operations from the main thread and thus improve the responsiveness of our application. In particular, we've seen how to export a large movie without blocking the playback of other movies that we might have open and without preventing the user from opening other movies. The basic rules we need to adhere to are relatively simple: (1) make sure to properly initialize and deinitialize the QuickTime environment on secondary threads (by calling EnterMoviesOnThread and ExitMoviesOnThread); (2) make sure to detach a movie from one thread and attach it to another thread if you need to operate on it in multiple threads (by calling DetachMovieFromCurrentThread and AttachMovieToCurrentThread); and (3) make sure to perform any user interface processing on the main thread.

Credits and References

A few of the routines used here are based on code by Michael B. Johnson. A more exhaustive discussion of threading in QuickTime can be found in Technical Note TN2125, "Thread-safe programming in QuickTime", available at http://developer.apple.com/technotes/tn/tn2125.html. You can also find a discussion of the QuickTime threading APIs in the QuickTime 6.4 API Reference.

About The Author

Tim Monroe is a member of the QuickTime engineering team at Apple. You can contact him at monroe@mactech.com. The views expressed here are not necessarily shared by his employer.

Mineralogy 101

Use The Quartz (2D), Luke!

By David Hill

Welcome!

While most of you have probably heard of Mac OS X's 2D drawing API, Quartz 2D, how many of you have taken the time to pull up the headers and get your hands dirty figuring out what it can do? For those of you that haven't, whether you've been too busy or just too intimidated to learn another drawing API, I'm going to walk you through some of the basics in this article. I'm not going to dwell on theory or the details of the drawing model. Instead, we're just going to play with the API and perhaps learn something along the way. I'll present short snippets of drawing code that you can drop into a sample project and try out. Feel free to play with the code and experiment on your own.

I'm basing the code on Mac OS X 10.3 (Panther) and Xcode 1.5 so if you're running a different version of the OS or Xcode, you'll need to adjust the content accordingly. In particular, watch out for functions that are only available in Panther. The Quartz 2D headers very specifically mark those APIs that are only available in 10.3 and later with the flag AVAILABLE_MAC_OS_X_VERSION_10_3_AND_LATER. In addition, while I'm providing a Cocoa project with this article, the Quartz 2D code is clearly marked and valid for Carbon as well as Cocoa. Carbon developers should watch out for differences between the coordinate systems, however, since Cocoa and Quartz 2D place the origin in the bottom left corner with +y pointing up while Carbon (older QuickDraw code as well as Carbon Events) assumes that the origin is in the upper left corner with +y pointing down.

One final note about Quartz 2D: the headers, functions, and types are all prefixed with CG as an abbreviation for Quartz 2D's older name, Core Graphics. Luckily, Xcode knows how to find the headers when you need to take a look. Just hit CMD-D, type the name of the header you're looking for ("CGContext.h", for example), and hit return. You can also CMD-double-click on a Quartz 2D type or function name in your code and Xcode will find the header for you.

The Quartz2DShell Project

Those of you following along at home will need a simple project to hold the Quartz 2D code snippets we'll be talking about. Either download the sample code for this article or start up Xcode and create a new Cocoa application project (for simplicity, I'm not using the Cocoa Document-based project). I've called mine Quartz2DShell but feel free to be more creative. Once Xcode has created the project, double click on the *MainMenu.nib* to open it in Interface Builder. Switch over to the Classes tab in the *MainMenu.nib* window, search for NSView, click on NSView in the class list, and select Subclass NSView from the Classes menu. IB should select the new NSView subclass (give it a suitable name like Quartz2DView). Then, select Create Files from the Classes menu and you're almost done. Add a Custom View to the window, resize it to fill the window, and set the resizing springs so that the view tracks the window. Finally, set the view's custom class to Quartz2DView and you're done. Save the nib file and return to Xcode. You should see *Quartz2DView.h* and *Quartz2DView.m* waiting for you.

Getting Started

The first thing you need in order to draw with Quartz 2D is a CGContextRef. For those of you familiar with other

drawing APIs, you can think of a CGContextRef as a QuickDraw port or a GDI device context. CGContextRefs hold your drawing state and give you a place to draw. There are several different ways to obtain a CGContextRef but for our purposes we'll simply retrieve one from our custom view by replacing the existing drawRect: method with the code shown below.

```
- (void)drawRect:(NSRect)rect
{
    // get the current CGContextRef for the view
    CGContextRef currentContext =
        (CGContextRef)[[NSGraphicsContext currentContext]
        graphicsPort];

    // grab some useful view size numbers
    NSRect bounds = [self bounds];
    float width = NSWidth( bounds );
    float height = NSHeight( bounds );
    float originX = NSMinX( bounds );
    float originY = NSMinY( bounds );
    float maxX = NSMaxX( bounds );
    float maxY = NSMaxY( bounds );
    float middleX = NSMidX( bounds );
    float middleY = NSMidY( bounds );

    ////
    ////  insert the Quartz 2D drawing code snippets here
    ////

    CGContextFlush( currentContext );
}
```

Carbon developers working with WindowRefs will need to get the port for the window and then call QDBeginCGContext() to obtain the appropriate CGContextRef. Note that you'll also need to call CGContextFlush() and then QDEndCGContext() once you're done drawing. Your function drawing code should look something like this:

```
CGrafPtr portPtr = GetWindowPort( windowRef );
CGContextRef currentContext = NULL;
QDBeginCGContext( portPtr, &currentContext );

// insert the Quartz 2D drawing code snippets here

CGContextFlush( currentContext );
// update the window
QDEndCGContext( portPtr, &currentContext );
```

Rectangles

As I mentioned earlier, the coordinate system of the CGContextRef has the origin in the bottom left corner and the +y axis points up. What I didn't tell you was that the context is also set up such that one context unit is equivalent to one view pixel. As we'll see a little later, Quartz 2D has a very flexible coordinate transformation system based on the concept of the Current Transformation Matrix (CTM) and so you can change that one-to-one mapping if you like. For now, we'll leave well enough alone and keep things simple. As long as we don't change the CTM, we can use coordinates that match the dimensions of the view and get the results we'd expect.

Let's start with some really basic drawing. Most of the Quartz 2D API is based on the concept of a path (we'll talk about paths in just a minute) but there is a really straightforward convenience function for drawing rectangles called CGContextFillRect(). CGContextFillRect() takes the CGContextRef you're drawing into and the rectangle you'd like Quartz 2D to fill. Insert the following code into the drawRect: method and then build and run the app.

```
    // try a simple rectangle with the convenience function
CGContextFillRect
    CGRect testRect =
        CGRectMake(  originX + width / 4.0,
    // left
                     originY + height / 4.0,
    // bottom
                     width / 2.0,
    // width
                     height / 2.0 );
    // height
    CGContextFillRect( currentContext, testRect );
```

You should see a large black rectangle in the middle of your window (Figure 1). Since we're basing our rectangle on the dimensions of the custom view, if you set up the resizing springs correctly in IB you should be able to resize the window and have the rectangle adjust accordingly in real time.

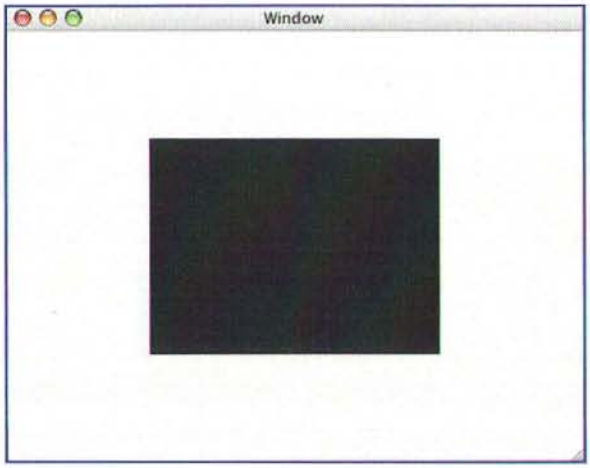


Figure 1: Black rectangles go with everything

Now that you've drawn your first Quartz 2D graphic, let's move on to a slightly more complicated topic. As I mentioned before, Quartz 2D likes to deal in paths. In fact, the CGContextFillRect() function really just creates, fills, and destroys a rectangular path under the hood. Let's take a look at what the CGContextFillRect() function is doing for us. Drop this code into the drawRect: code in place of the previous snippet:

```
CGContextBeginPath( currentContext );
CGContextMoveToPoint( currentContext,
    originX + width / 4.0, originY + height / 4.0 );
CGContextAddLineToPoint( currentContext,
    originX + width / 4.0 + width / 2.0,
    originY + height / 4.0 );
```

```
CGContextAddLineToPoint( currentContext,
  originX + width / 4.0 + width / 2.0,
  originY + height / 4.0 + height / 2.0 );
CGContextAddLineToPoint( currentContext,
  originX + width / 4.0,
  originY + height / 4.0 + height / 2.0 );
CGContextFillPath( currentContext );
```

Here, we're starting a new path with CGContextBeginPath(), moving to the bottom left corner of the rectangle, adding lines along the bottom, right, and top edges, and then filling the path with CGContextFillPath(). Note that we didn't have to add the fourth line to close the path ourselves. For the purposes of filling the path, Quartz 2D will automatically close the current path (if it is still open) with a straight line back to its starting point whenever we call one of the drawing functions.

One interesting thing to note about Quartz 2D and paths: drawing with the current path (either filling, stroking, or both) consumes that path. If we were to add another drawing call like CGContextStrokePath() after our call to CGContextFillPath(), nothing would happen. We would have to recreate our rectangular path in order to draw it again. Thankfully, since performing both a fill and a stroke (drawing the outline) of a path is so common, Quartz 2D provides a way to do both in one function call. Simply call CGContextDrawPath() and pass in kCGPathFillStroke or kCGPathEOFillStroke for the drawing mode parameter and Quartz 2D will fill and stroke the path in one operation.

In order for the stroke to be visible over the fill, I'll need to show you one more thing: colors. There are two important colors in Quartz 2D: the fill color and the stroke color. There are several ways to set colors but, for the purposes of this article, we'll stick with the simplest two: CGContextSetRGBFillColor() and CGContextSetRGBStrokeColor(). (If you want to investigate Quartz 2D's color support in more detail, check out CGColorRefs and CGColorSpaceRefs). Replace the call to CGContextFillPath() with these lines and watch what happens:

```
// Set the red, green, blue, and alpha color values
CGContextSetRGBFillColor( currentContext,
  1.0, 0.0, 0.0, 1.0 );
CGContextSetRGBStrokeColor( currentContext,
  0.0, 1.0, 0.0, 1.0 );
CGContextDrawPath( currentContext, kCGPathFillStroke );
```

Now your code should draw a red rectangle with a green outline. For those of you like me with poor eyesight, a bad monitor, or both, you may have some trouble seeing the single-pixel outline. Let's make the stroke wider and more obvious. Add this line before the call to CGContextDrawPath():

```
CGContextSetLineWidth( currentContext, 10 );
```

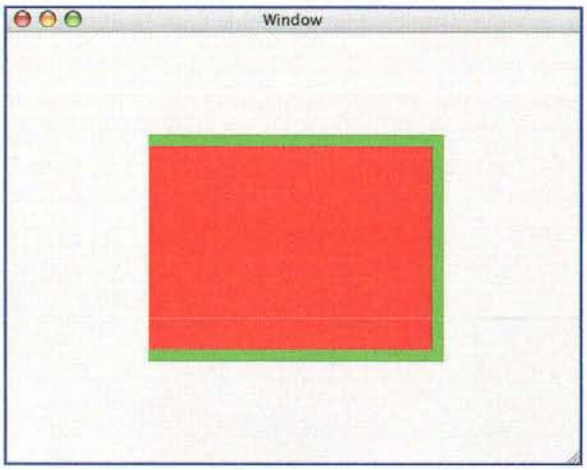


Figure 2: Looks like we forgot something

Ah, now that the outline is more visible (Figure 2), you can clearly see why I said that Quartz 2D would close the path for the purposes of ***filling*** the path. Quartz 2D didn't automatically close the path for the stroke so the left edge of the rectangle doesn't have a green line. Close the path manually (Figure 3) by adding this call to CGContextClosePath() after the third CGContextAddLineToPoint() call:

```
CGContextClosePath( currentContext );
```

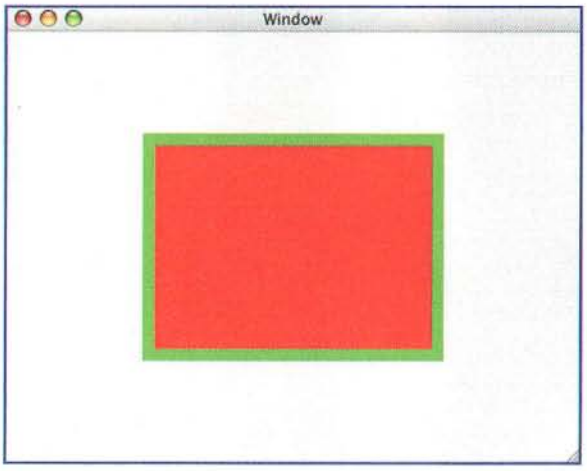


Figure 3: That's much better

Lines

Now that you've got a handle on rectangles, we'll move on to some of the fun things you can do with lines. We've already added lines to paths to make our rectangles and changed the line width but there are some other things we can tweak to control the way lines look. Replace your rectangle code with the line code below that explores the different line cap styles.

```
// draw some wide lines with different endcaps and dashes
CGContextSetLineWidth( currentContext, 15 );

CGContextSetLineCap( currentContext, kCGLineCapButt );
CGContextBeginPath( currentContext );
CGContextMoveToPoint( currentContext,
  originX + 15, originY + 15 );
CGContextAddLineToPoint( currentContext,
  maxX - 15, originY + 15 );
CGContextStrokePath( currentContext );

CGContextSetLineCap( currentContext, kCGLineCapRound );
CGContextBeginPath( currentContext );
CGContextMoveToPoint( currentContext,
  originX + 15, originY + 45 );
CGContextAddLineToPoint( currentContext,
  maxX - 15, originY + 45 );
CGContextStrokePath( currentContext );

CGContextSetLineCap( currentContext, kCGLineCapSquare );
CGContextBeginPath( currentContext );
CGContextMoveToPoint( currentContext,
  originX + 15, originY + 75 );
CGContextAddLineToPoint( currentContext,
  maxX - 15, originY + 75 );
CGContextStrokePath( currentContext );
```

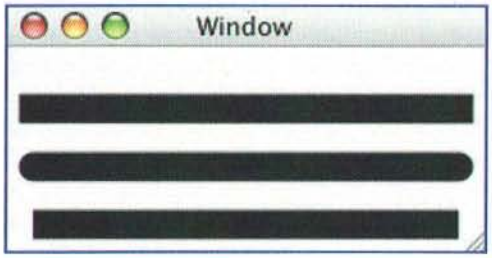


Figure 4: Line cap styles

This code produces three fat lines (Figure 4) with different end styles: butt, round, and square. Feel free to increase the line size or zoom in (using Mac OS X's accessibility controls) to get a better look at the end caps. Also, notice that while the line with the kCGLineCapButt style ends right where the path starts and ends, the other two line cap styles cause the line to overflow the end points. This happens because the round and square cap styles draw a half-circle and a square, respectively, centered on the end point of the path whereas the butt cap style squares off the end of the path perpendicular to the path at the endpoint.

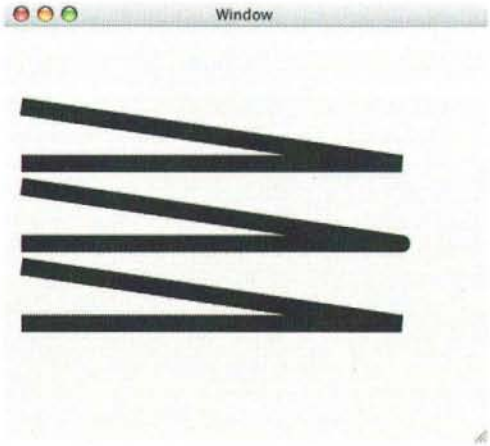


Figure 5: Line join styles

Quartz 2D can also join lines in three different ways: miter, round, and bevel. The following code draws three paths composed of two lines each (Figure 5). Notice the difference in the corners where the lines meet. Depending on the size of your window, the miter and bevel join styles may look the same. Narrow the window slowly and the mitered lines will eventually change as the angle between the lines increases (Figure 6). I don't know about you but I'm really glad Quartz 2D figures all of this out for me.

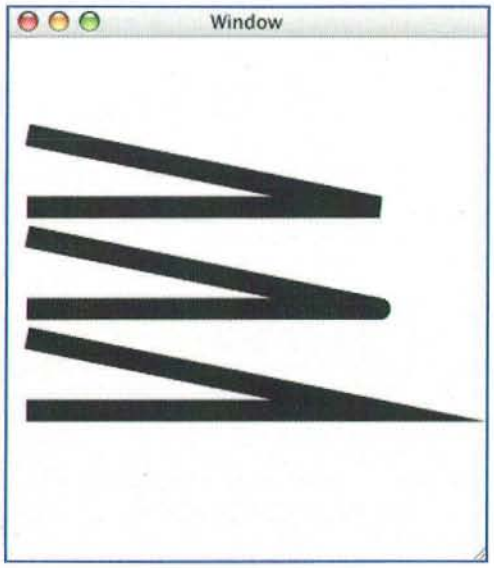


Figure 6: The miter join looks different now

```
// test the different line join styles
CGContextSetLineWidth( currentContext, 15 );
CGContextSetLineCap( currentContext, kCGLineCapButt );

CGContextSetLineJoin( currentContext, kCGLineJoinMiter );
CGContextBeginPath( currentContext );
CGContextMoveToPoint( currentContext,
  originX + 15, originY + 105 );
CGContextAddLineToPoint( currentContext,
  maxX - 75, originY + 105 );
CGContextAddLineToPoint( currentContext,
  originX + 15, originY + 155 );
CGContextStrokePath( currentContext );

CGContextSetLineJoin( currentContext, kCGLineJoinRound );
CGContextBeginPath( currentContext );
CGContextMoveToPoint( currentContext,
  originX + 15, originY + 175 );
CGContextAddLineToPoint( currentContext,
  maxX - 75, originY + 175 );
CGContextAddLineToPoint( currentContext,
  originX + 15, originY + 225 );
CGContextStrokePath( currentContext );

CGContextSetLineJoin( currentContext, kCGLineJoinBevel );
CGContextBeginPath( currentContext );
CGContextMoveToPoint( currentContext,
  originX + 15, originY + 245 );
CGContextAddLineToPoint( currentContext,
  maxX - 75, originY + 245 );
CGContextAddLineToPoint( currentContext,
  originX + 15, originY + 295 );
CGContextStrokePath( currentContext );
```

Paths

Up to this point, we've been drawing by setting up the current path in a context and then asking the context to draw that path. This works fine but it can be awkward when you want to reuse a path multiple times. Luckily, Quartz 2D includes a separate path object, CGPathRef, that you can create and reuse over and over again. The usage pattern is simple: create a mutable path, add to the path in much the same way as we did for the current path in the context, add the path to the context, and ask the context to draw. Note: the NULL parameters below are CGAffineTransforms that allow you to transform the points as they're added to the path. We won't worry about those in this article so we'll pass in NULL to keep Quartz 2D from doing any extra work. Drop in the code snippet below and try it out.

```
// create a path and use it
path = CGPathCreateMutable();
CGPathMoveToPoint( path, NULL,
  originX + 15, originY + 120 );
CGPathAddLineToPoint( path, NULL,
  maxX - 75, originY + 120 );
CGPathAddLineToPoint( path, NULL,
  originX + 15, originY + 170 );
CGContextAddPath( currentContext, path );
CGPathRelease( path );
CGContextStrokePath( currentContext );
```

Not too exciting, is it? Have patience. We're getting to the good part. Creating a CGPathRef, adding to it, drawing with it, and then releasing it isn't terribly efficient. What you really want to do is create the path, set it up, and then use it multiple times before releasing it. Get rid of the boring path code above and replace it with this:

```
CGContextSetLineWidth( currentContext, 2 );

// create and set up the path starting at the origin
path = CGPathCreateMutable();
CGPathMoveToPoint( path, NULL, originX, originY );
CGPathAddLineToPoint( path, NULL,
  originX + width / 8.0, originY );
CGPathMoveToPoint( path, NULL,
  originX + width / 8.0, originY + 0.25 * width / 8.0 );
CGPathAddLineToPoint( path, NULL,
  originX + 0.75 * width / 8.0,
  originY - 0.25 * width / 8.0 );

// translate the origin to the middle of the window
CGContextTranslateCTM( currentContext,
  middleX, middleY );

// scale the context so that things are twice as big in both directions
CGContextScaleCTM( currentContext, 2, 2 );

float radiansToRotate = 2 * M_PI / 21;
int i;
for ( i = 0; i < 21; i++ )
{
  // add the path to the context and draw
  CGContextAddPath( currentContext, path );
  CGContextStrokePath( currentContext );

  // rotate the context just a bit
  CGContextRotateCTM( currentContext, radiansToRotate );
}

// we're done with the path
CGPathRelease( path );
```

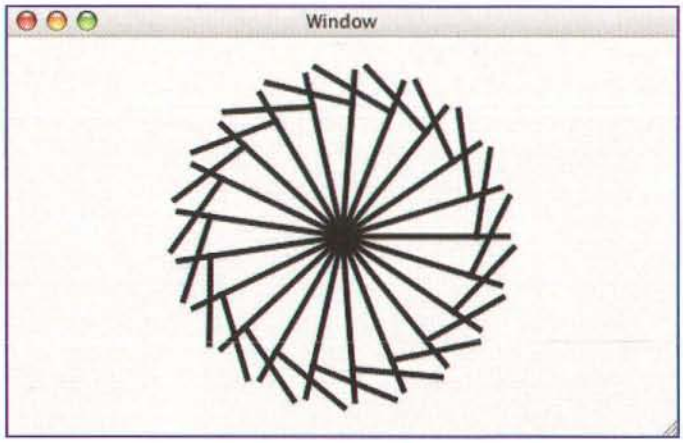


Figure 7: Repeated and rotated path

This code adds some new twists (Figure 7) to what you've seen before by manipulating the CTM. Remember the CTM? I told you that as long as we didn't mess with it, we could draw using view coordinates and things would turn out like we'd expect. Well, now we've got to change the CTM or we won't be able to see that we're drawing the CGPathRef object multiple times. In this case, we set up the path, set up the CTM so that the coordinate system origin is in the middle of the window, make things a bit bigger, and then enter a loop. The loop draws the path object and then rotates the CTM a bit so that we don't draw the path over itself 21 times in a row. Note: CGContextRotateCTM() rotates the CTM such that the next path you draw appears to rotate counter-clockwise around the origin.

There's one last stop on our tour of paths and lines: dashes. Quartz 2D makes it almost too simple to draw beautiful dashed lines. All you need to do is set up an array containing the dash lengths you want, make a call to CGContextSetLineDash(), and stroke your paths. Quartz 2D takes care of the rest. Take this code for a spin (Figure 8) and see what you think.

```
// play with line dashes
CGContextSetLineWidth( currentContext, 10.0 );
float dashPhase = 0.0;
float dashLengths[] = { 20, 30, 40, 30, 20, 10 };
CGContextSetLineDash( currentContext,
  dashPhase, dashLengths,
  sizeof( dashLengths ) / sizeof( float ) );

CGContextMoveToPoint( currentContext,
  originX + 10, middleY );
CGContextAddLineToPoint( currentContext,
  maxX - 10, middleY );
CGContextStrokePath( currentContext );
```

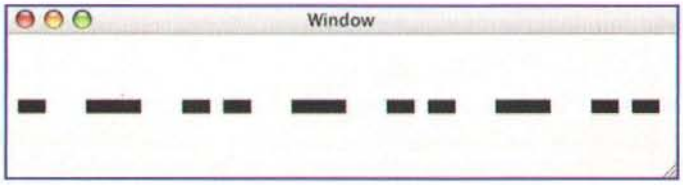


Figure 8: A custom dashed line

Here, the dashLength[] array sets up dashes of length of 20, 40, and 20 with gaps in between of length 30, 30, and 10. You can also adjust the phase (where the dash pattern starts - play with it a bit) when you call CGContextSetLineDash().

Dashes work fine with straight, horizontal lines but that's boring. Let's try something more interesting. Here's another snippet that draws curves instead of lines (Figure 9). Add this code after the CGContextStrokePath():

```
// play with curves & dashes
CGContextSetRGBFillColor( currentContext,
  1.0, 0.0, 0.0, 1.0 );
CGContextSetRGBStrokeColor( currentContext,
  0.0, 1.0, 0.0, 1.0 );

CGContextMoveToPoint( currentContext, middleX, maxY );
CGContextAddCurveToPoint( currentContext, originX, maxY,
  originX, 0.667 * maxY, middleX, middleY );
CGContextAddCurveToPoint( currentContext,
  maxX, 0.333 * maxY, maxX, originY, middleX, originY );
CGContextAddCurveToPoint( currentContext, originX,
  originY, originX, 0.333 * maxY, middleX, middleY );
CGContextAddCurveToPoint( currentContext, maxX,
  0.667 * maxY, maxX, maxY, middleX, maxY );

CGContextDrawPath( currentContext, kCGPathFillStroke );
```

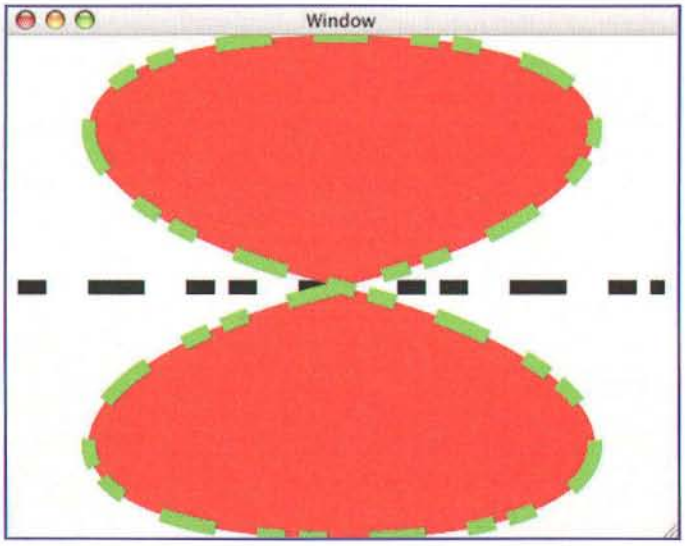


Figure 9: Dashed and filled figure eight

Notice how the dash pattern moves along the curve as you resize the window while the pattern remains fixed on the horizontal line? That is because Quartz 2D starts the dash pattern at the beginning of the path and repeats the pattern as many times as necessary to reach the end of the path. If the length of the path changes, Quartz 2D just continues the pattern as necessary. Since we're always looking at the beginning of the path for the straight line, the pattern doesn't appear to shift. However, with the curve we can see a clear beginning and end of the path. That makes the additional dashes added and removed quite obvious as the curve length changes.

Shadows

Mac OS X has always provided very nice shadows under windows and other UI elements. With Panther, Apple added APIs to Quartz 2D to allow you to draw your own shadows. There are two shadow APIs: the simple one and the slightly less simple one. The former, CGContextSetShadow(), sets up a simple black shadow. The latter, CGContextSetShadowWithColor(), allows you to specify the color of the shadow. We'll use the black version here but knock yourself out with colored shadows if that's your thing. Add this line right after the "play with curves & dashes" comment.

```
CGContextSetShadow( currentContext,
  CGSizeMake( 14.0, -14.0 ), 7.0 );
```

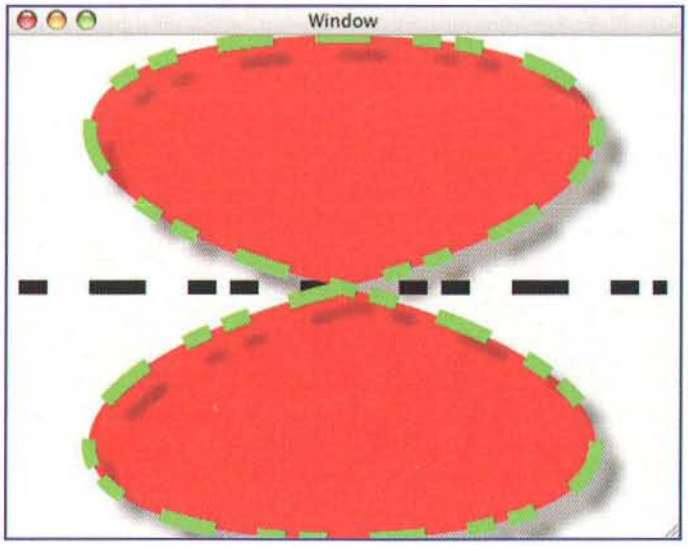


Figure 10: New and improved with Shadows!

Now our curve has a nice soft shadow (Figure 10) but if you look closely you'll see that the dashed green outline is casting a shadow on the red filled portion. Depending on the look you're going for, that may or may not be what you want. It turns out that even though we're making a single call to fill and stroke the path, Quartz 2D is performing two separate operations and each result has a shadow. The fill comes first with the bottom shadow and then the stroke comes along and draws its shadow on top of the fill. So, if you want the combined result of the two operations to have a single shadow, what do you do? You'll need to use a transparency layer, yet another new feature added to Quartz 2D for Panther.

Transparency layers only require two function calls, one to begin the layer and one to end the layer. In between, all Quartz 2D drawing for the context goes into the layer instead of the destination of the context. Once you end the layer, all of the drawing in the layer is composited into the context using the shadow state of the context. It will make more sense if you try it out yourself. There are two lines of code below. Add the CGContextBeginTransparencyLayer() call right after the call to CGContextSetRGBStrokeColor() and the CGContextEndTransparencyLayer() call right after the CGContextDrawPath() call.

```
CGContextBeginTransparencyLayer( currentContext, NULL );
  CGContextEndTransparencyLayer( currentContext );
```

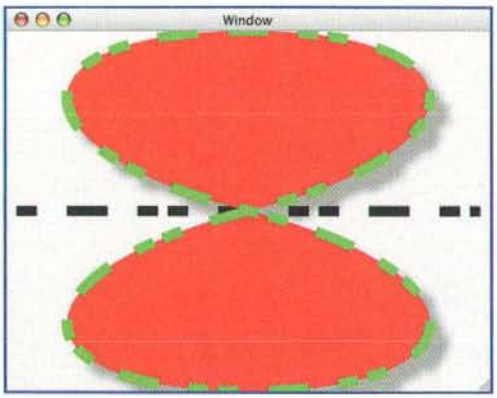


Figure 11: Just one shadow this time

With that change, you should see your green-outlined, red-filled figure-eight with a single shadow behind the whole thing (Figure 11). Oh, and you're probably still drawing the black dashed line in the background. Transparency layers make it very easy to add shadows to whole groups of objects. As their name implies, you can also use transparency layers to properly handle alpha compositing (transparency) of multiple objects at the same time by setting the alpha value for the context before you begin the transparency layer. Call CGContextSetAlpha() before you call CGContextBeginTransparencyLayer() and the contents of the layer will be composited back into the context with that alpha value when you end the layer. To see this in action, add the following call right before you begin the layer:

```
CGContextSetAlpha( currentContext, 0.5 );
```

Now the whole figure eight has faded out and you can see some of the black dashed line running through the middle (Figure 12). You can also see the faint outline of the figure's shadow through the red fill on the left side.

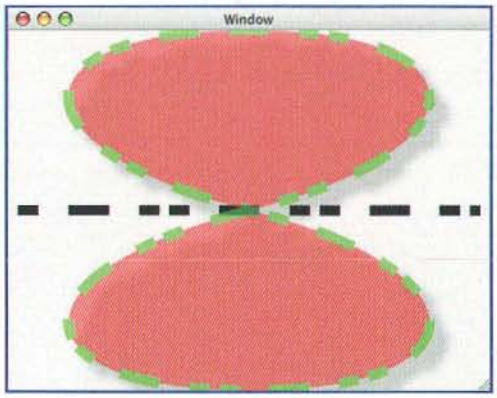


Figure 12: Fading out

Wrap Up

I hope you've enjoyed our brief exploration of Quartz 2D. There's so much power and elegance in the API, you really need to sit down and play with it for a while before you get a feel for what it can do. With any luck, I've given you enough of a taste of Quartz 2D that you're ready to go off and experiment on your own. Enjoy!

About The Author

David Hill is a freelance writer living in College Station, Texas. In a former life, he worked in Apple's Developer Technical Support group helping developers print, draw, and write games. In his free time he dabbles in screen savers and other esoteric topics.

DEV DEPOT

DevDepot has it all!

Get More out of your Mac!

Visit our online store today for special offers and great new products.
www.devdepot.com

KEYSPAN
Digital Media Remote

ONLY $39.99

Control your computer's media programs just like your TV!

For iTunes, Windows Media Player, DVD, CD, and more!

iTripmini
FM Transmitter

ONLY $39.99

Listen in your car!

Designed exclusively for the iPod mini.

NO BATTERIES NEEDED!

iTalk
iPod Voice Recorder

ONLY $34.99

Works as a loud speaker!

Turn your iPod into a world-class voice recorder.

SightLight
FireWire light for iSight

$37.99

AC Adapter
For Powerbook and iBook laptops

$39.99

Keypoint Remote
Multimedia RF remote control

$49.99

Noise Reduction Headphones

$59.99

FlashDrive
Cyclops 256 MB USB Flash Drive

$79.99

PowerWave
USB Audio Interface & Amplifier

$89.99

PowerMate
USB Multimedia Controller & Input Device

$36.99

iceKey
Slim USB Keboard

$44.99

iMic
USB Audio Interface

ONLY $34.99

A must-have device for people who are serious about high quality audio.

Connect virtually any sound device!

TechTool Pro 4.0
for Mac OS X

The ultimate emergency software is now available for OS X!

$89.99

UNIX Utilities
for Mac OS X 3.0

Everything you need to turn your Mac into a UNIX Workstation.

$39.99

Office Applications
for Mac OS X 3.0

Packed with office and productivity apps!

$29.99

Office Applications
for Mac OS X 3.0

Bug reporting and organizing!

* Includes free bug reporter with each application release!

$79.99

DEV DEPOT

www.devdepot.com

Nuts About SquirrelMail

Walking into MacWorld Expo at the Moscone Center in San Francisco with an Exhibitor's badge the day before the Expo opens is one of the strangest experiences to be had. There's the sound of duct tape whizzing off of rolls, and the slap of mats falling into place, accompanied by the hum of electrical equipment. It's a great chance to walk around and get acclimated to the lay of the land before the crowds pile in.

Away from the Office

Sooner or later, though, the urge to check email hits, and then it becomes quite apparent that even though the booths are partially set up, there's no open Airport networks available, and without a password, no way to check email. But then, towards the back of Moscone, there's a few iMac G5s setup for an "Internet café" and there's someone sitting there feverishly typing away. After testing some of the other iMacs, it's obvious that the only one that works is the one ***he's using***, and sitting there politely isn't going to help, as if it were the last public phone on the planet (which it is, sort of). What makes it even worse is that he's using SquirrelMail, and judging by the number of folders and subfolders in the side bar, it's obvious that he's an email nutball, and that he's never going to stop, since he's got access to everything he's every received, sent, and squirreled away.

Every day I hear more and more instances of what I call the "great SMTP (Simple Mail Transfer Protocol) lockdown," where ISPs such as Earthlink, Comcast, RCN and others are blocking all traffic over TCP port 25, which means that it's become increasingly difficult to use a corporate email account from home, without either setting up an alternate SMTP port or the ISP's own SMTP server. Not too long ago, all a home broadband user had to do was use authenticated SMTP, but these days even that's not a ***fait accompli***. The ISPs claim to be blocking port 25 in the name of spam prevention (and they usually do it ***without notifying their customers first***), but it's pretty obvious that they want to lock you into using ***their*** email servers and email addresses, which is the easiest way they have of insuring that you won't switch to another ISP with a better deal or faster service. My girlfriend, for example, switched to Covad DSL service over a year ago, yet still pays $14.95 a month for Yahoo dial-up service, just so she can "save" her email address. It's a scam on a grand scale—even if you're savvy enough to register your ***own*** domain name, and purchase email services or even run your own email server, you might not be able to easily send mail via that account.

In these days of predatory if your broadband provider decides to block it. Where web browsing speeds might be accelerating at a breakneck pace, but where sending email through any server except for the ISP's is heavily deprecated, it's webmail that's becoming an absolutely critical service for corporate email users when they're out of the office. After all, which ISP would dare to block TCP port 80, through which all basic World Wide Web traffic travels? Although high-end groupware servers like Microsoft Exchange and Lotus Notes have featured webmail since the mid 1990s, other entities like Google are getting heavily into webmail services, taking advantage of their well-known brand names and their ISP-independent status

to lure email users to park their email addresses at the gmail.com domain for the long term (with wickedly fast searches and two gigabytes of storage), so that account holders may switch ISPs with impunity as they move from location to location or find better deals on broadband, while being exposed to more and more of Google's pay-per-click ads, which have now become both the ubiquitous marginalia and tip jars of web sites.

So, where once web-based email was the tool that was *occasionally* used to send and receive email while away from the office when a corporate laptop wasn't available, it's now become a standard arrow in the corporate quiver—often the lifeline of communication between remote email users and the main office when standard email server is blocked, or when a VPN is too unwieldy or when corporate IT help desk staffers want to draw a line in the sand when it comes to supporting their employees' home networks.

Highly Trained Squirrels

Roald Dahl's *Charlie and the Chocolate Factory* featured a room full of specially trained Squirrels that tested each nut that went into Wonka candy bars, then tossed out the bad nuts and shelled the good. In the same vein, it seems that Apple's OS X Server Development team coveted trained squirrels as well! The squirrels I'm speaking of are the hard-working rodents of the SquirrelMail project, an open-source PHP (Pre HyperText Processor) script living at http://squirrelmail.sourceforge.net that leverages the Apache web server to provide a powerful and extensible webmail service. The SquirrelMail team describes the project as:

> . . . a standards-based webmail package written in PHP4. It includes built-in pure PHP support for the IMAP and SMTP protocols, and all pages render in pure HTML 4.0 (with no JavaScript required) for maximum compatibility across browsers. It has very few requirements and is very easy to configure and install. SquirrelMail has all the functionality you would want from an email client, including strong MIME support, address books, and folder manipulation.

From its inception, SquirrelMail was designed with the notion of "install once, access with any browser, on any platform" in mind, which is a great compliment to the basic premise of web-based email, where what operating system and web browser the user is running, and on what speed and type of network (and with what ports blocked) makes little or no difference. Install SquirrelMail, configure it, and then let the remote users do the rest themselves. It's a thing of beauty. Given that Apache and PHP come pre-installed on nearly all UNIXy operating systems, it's no surprise that SquirrelMail has become the darling of the Linux, BSD, and OS X worlds, and quite simply, one of the most consistently popular projects at Sourceforge.net for a number of years, like Pink Floyd's "Dark Side of the Moon," it's always hanging near the top of the charts.

Figure 1. Highly Trained Squirrels Sort Your Mail for You

However, it seems that the OS X Server team wasn't exactly *proud* to employ the little varmint, so they tried their best to hide the squirrel behind the scenes, a charade that's easily foiled by a peek into the terminal (or just the OS X Server Web Services Documentation). Like many of the open-source projects included with OS X Server, there's an amount of obfuscation to cover up the origins of the goodies, at least at the GUI layer, but once logged into SquirrelMail, regardless of whether the logo's an actual squirrel or postage stamp, there's no mistaking the interface!

Figure 2. "Squirrels? What Squirrels?" OS X Server Webmail Logo

Getting Going (and Along) with OS X Server

Enabling SquirrelMail on OS X Server is a trifle, simply edit the web site settings in Server Admin, and check the "webmail" box in the options list. Email services must be running and configured, along with user accounts that have email boxes and IMAP (Internet Mail Access Protocol) enabled. IMAP is a requirement for SquirrelMail (and all webmail programs designed with the sanity of the administrator in mind). Using IMAP means that all messages

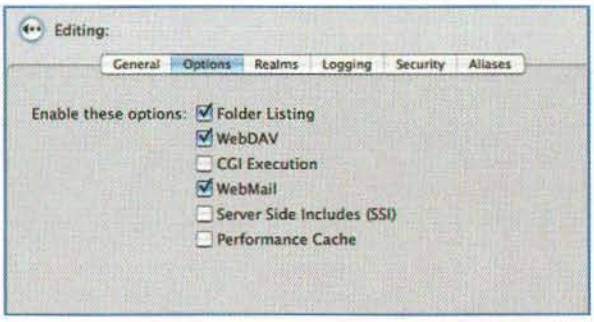


Figure 3. Checkbox to Enable WebMail in Server Admin for Tiger Server

remain on the server, and are read on the server, with only the HTML representation of their content actually transmitted back to the webmail users. If no attachments are involved, email displays via the web browser as quickly as the server can process the messages and generate the HTML (dependent, of course, on the connection speed of the user logging in). Once enabled, all the user has to do is navigate to http://yourwebsitename/webmail, and then login at the prompt:

Figure 4. Http://yoursite.com/webmail

Although SquirrelMail has been bundled with OS X Server starting with version 10.2, it really started to kick some furry tail with OS X Server 10.3 and 10.4, which use the Cyrus IMAP server, rather than the Apple Mail Server Carbonized from AppleShare IP that OS X Server 10.1 and 10.2 featured. With Cyrus, SquirrelMail absolutely ***flies***, and with a decent broadband connection, can be even faster than most ***non-web mail clients***, such as Mail.app or Microsoft Entourage.

The Squirrel Behind the Curtain (a.k.a Road Kill)

Normally, SquirrelMail would be installed into a single directory as a separate virtual domain on a web server, so that the users would go to http://webmail.yourdomain.com, rather than http://www.yourdomain.com/webmail. Apple's implementation, however, makes it much easier to leverage a single copy of SquirrelMail over multiple sites. Finding the "parts" of the SquirrelMail installation isn't exactly straightforward. For example, the URL would suggest that SquirrelMail was installed in a subfolder of the webroot /Library/WebServer/Documents, but it isn't! It's almost as if OS X Server ***flattened*** the poor PHP script, scattering its code to the far corners of the OS X Server filesystem. One interesting feature of Tiger Server is that when webmail is enabled in the default web site, it's now indicated via an "Available services:" box under the Tiger Server Logo, which serves as a hyperlink to SquirrelMail:

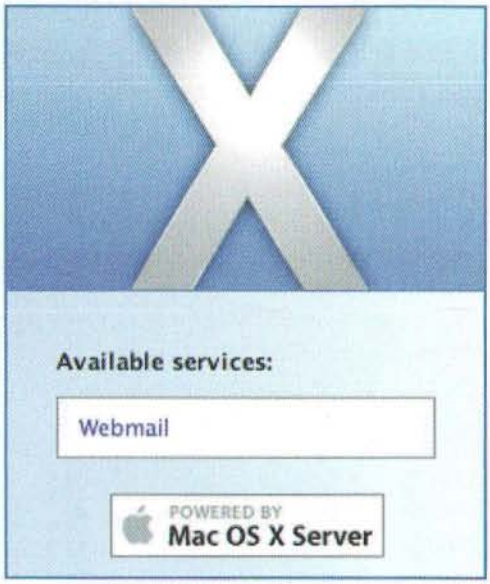


Figure 5. Webmail link on Tiger Default Web Page

Apple's implementation of SquirrelMail starts in the /etc/httpd directory with the rather self-explanatory httpd_squirrelmail.conf file:

```
nagitest:/etc/httpd mostadmin$ cat
httpd_squirrelmail.conf
# Config file for linking SquirrelMail to MacOSX Server
Web Server virtual host.
# Add the following line to each virtual host for which
you want SquirrelMail:
#    Include /etc/httpd/httpd_squirrelmail.conf
# Browsers will then be able to reference SquirrelMail
with a URL like
#    http://virtualhost.example.com/WebMail/
Alias /WebMail /usr/share/squirrelmail
Alias /webmail /usr/share/squirrelmail

<Directory /usr/share/squirrelmail>
  Options Indexes FollowSymLinks
</Directory>
```

The crumb trail leads to /usr/share/squirrelmail, where the bulk of what a "normal" SquirrelMail installation lives. The first thing to do is replace the postage stamp logo with the company, club, user group, or house of worship logo. To do so, simply prepare the logo in one of the supported formats (.png, .jpg. .gif), and copy it to the /usr/share/squirrelmail/images directory. Next, it's time to fire up the SquirrelMail configure script:

```
nagitest:/usr/share/squirrelmail mostadmin$ sudo sh
configure

SquirrelMail Configuration : Read: config.php (1.4.0)
---------------------------------------------------------
Main Menu --
1.  Organization Preferences
2.  Server Settings
3.  Folder Defaults
4.  General Options
5.  Themes
6.  Address Books
7.  Message of the Day (MOTD)
8.  Plugins
9.  Database
10. Languages

D.  Set pre-defined settings for specific IMAP servers

C   Turn color on
```

```
S    Save data
Q    Quit

Command >> 1
```

This menu-driven configuration script is both simple and powerful. It's somewhat reminiscent of the pre-web Internet, where a cousin of the Squirrel, known as Gopher, served up information via text-driven menus. Changing the logo on the webmail login page is as simple as choosing "Organization Preferences" from the menu and typing in the path to the new image:

```
SquirrelMail Configuration : Read: config.php (1.4.0)
---------------------------------------------------
Organization Preferences
1.   Organization Name        : Mac OS X Server WebMail
2.   Organization Logo        : mostlogo.gif
3.   Org. Logo Width/Height   : (0/0)
4.   Organization Title       : SquirrelMail $version
5.   Signout Page             :
6.   Top Frame                : _top
7.   Provider link            :
http://www.squirrelmail.org/
8.   Provider name            : SquirrelMail

R    Return to Main Menu
C    Turn color on
S    Save data
Q    Quit

Command >>
```

Figure 6. Webmail login with custom logo

To cover every single configuration option in SquirrelMail would take a very long time, but it's interesting to note that the Configure script kicks off a perl script which lives in the /usr/share/squirrelmail/config directory called conf.pl. Executing conf.pl (use **sudo perl conf.pl**) from the command line brings up the familiar SquirrelMail configuration menu, but adds a little extra hint when exiting:

```
Exiting conf.pl.
You might want to test your configuration by browsing to
http://your-squirrelmail-location/src/configtest.php
Happy SquirrelMailing!
```

While the configtest.php feature is nice, it also can be somewhat of a security risk, since it reveals details about the server configuration, notably regarding the authentication settings of the IMAP server. Another very important directory to administering SquirrelMail is /var/db/squirrelmail/ and its two subfolders, attachments and data. The attachments folder contains mine-encoded files send with email messages:

```
nagitest:/var/db/squirrelmail/attachments root# ls -al
-rw----    1 www    wheel   2614916 Jul 26 23:55 NUSgHDdB6qiMQr0eO6CE3MPixlSXgSJB
-rw----    1 www    wheel   7947926 Jul 27 00:05 XHAca4OO4T2xzloNZCzo5C3UznWAolY1
-rw----    1 www    wheel   4082991 Jul 26 23:58 bJEprLCFiVQKrY6QOu08ZtHKxWp5hfDr
```

It might not hurt to check this directory and periodically clean out the contents to save some disk space. It may even be worthwhile to script a small cron or launchd job to periodically sweep away the SquirrelMail droppings. In a similar vein, if a particular user experiences anomalies in the display of their webmail, strange error messages, or long delays while logging in, it might be worthwhile to clean up an session files, or delete the users' .pref files in the data folder. For the most part, the out-of-the-box SquirrelMail configuration for OS X Server is pretty good, but can be made a lot better with just a few more tweaks.

Adjusting Authentication

The first thing to do is to review how the SquirrelMail web interface sends its authentication credentials to the imap server, and while there's technically no risk because the out-of-the-box configuration has the IMAP server and the webmail server on the same box, the login credentials aren't sent over the Internet. However, I still prefer SSL security for webmail, though "rolling your own" SSL certificates is a topic for another column. It's mandatory if using SquirrelMail to access an IMAP account on a remote server, which I do, because I like to run my own "Personal" copy of SquirrelMail on my own G5, and sometimes my company provides webmail services for customers who don't want or can't maintain their own webmail server for whatever reason. Also, disabling the login and plain authentication types for Mail Services in OS X Server 10.4 requires that SquirrelMail also be configured for an allowed authentication type, in this case cram-md5, which while not as secure as SSL, is certainly much better than plain and login authentication, still used by a surprising number of mail servers. First, the email service needs to be configured. Authentication types are located in Server Admin, Mail Service, Settings, Advanced:

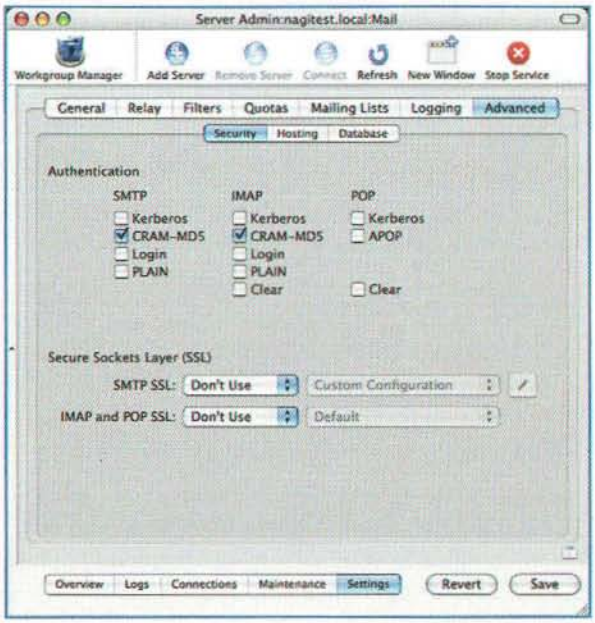


Figure 7. Disabling Insecure IMAP Protocols

The next step is to adjust the authentication type in SquirrelMail's configure script, and that's done in the IMAP server settings section of the Server Settings menu. Simple.

```
IMAP Settings
--------------

4.  IMAP Server              : localhost
5.  IMAP Port                : 143
6.  Authentication type      : cram-md5
7.  Secure IMAP (TLS)        : false
8.  Server software          : cyrus
```

Increasing the Attachment Size

Another out-of-the-box limitation is one imposed by the PHP library itself: SquirrelMail on OS X Server can only handle email attachments of only ***two megabytes or less***. While an Xserve running OS X Server 10.3 or 10.4 with a gig of RAM and an Xserve RAID attached by fiber channel makes for a ***flying SquirrelMail***, the limit's simply the PHP default and with a few small edits, can be made to match to the attachment size limit of the Mail server itself. However, I have to point in all fairness out that the user experience of uploading a twenty-five megabyte attachment through a web browser might be somewhat different than what end-users expect. As a matter of fact, I've found that some end users don't adjust their expectations when working at home over a cable modem that might download at similar speeds to what they're used to at the office, but with much slower upload speeds. Even worse, I've had complaints of SquirrelMail "not working" simply because someone didn't understand how long it would take to upload a ten megabyte PowerPoint presentation via a dial-up connection.

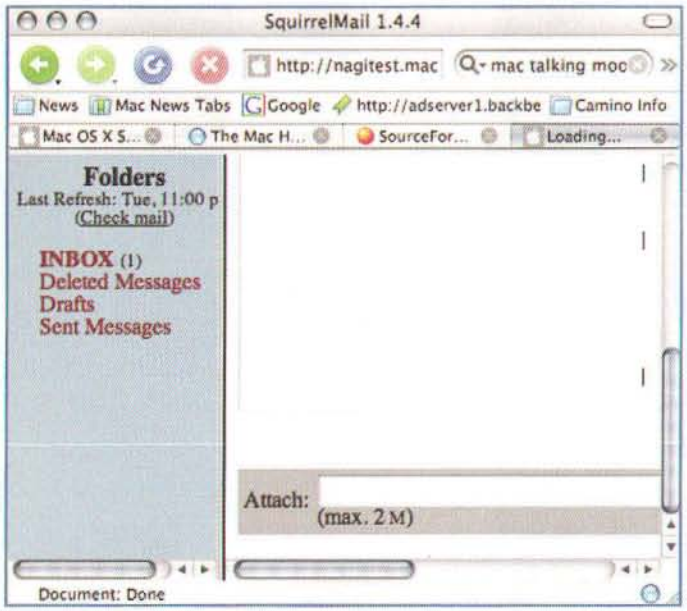


Figure 8. SquirrelMail default attachment size

Uploading large files via a web browser on OS X doesn't really offer much in the way of a progress indicator, and if the connection's slow (like dialup) or flaky (like dialup over a Bluetooth modem connection), then there's always the risk that the SquirrelMail user might be left in limbo. But if they're forewarned, upping the attachment size can even make use of Cyrus's ability to act as a file system, much like folks would use a .mac account or a gmail account to Squirrel away or transfer files.

The first surprise is that the PHP distribution shipped with Tiger Server doesn't include an active copy of the PHP.ini file, the standard PHP configuration file. Luckily, there's an /etc/php.ini.default file that can be copied and edited to spec. First, make a copy of the file:

```
nagitest:/etc dean$ sudo cp /etc/php.ini.default
/etc/php.ini
```

then, open up the php.ini file in a favorite editor. These days I'm partial to BBEdit's command line "bbedit" tool, or TextWrangler's "edit" tool, but pico or vi will also work just fine. It's helpful to have something with an easy search feature:

```
nagitest:/etc dean$ sudo pico php.ini
```

Now, find the first thing to adjust, the following line:

```
max_execution_time = 30 to max_execution_time = 600
```

this allows for longer uploads (ten minutes) without causing a timeout. Next, adjust the line that reads:

```
memory_limit = 8m to read memory_limit = 32M
```

Next is the line:

```
post_max_size = 2M to post_max_size = 24M
```

Now the line:

```
upload_max_filesize = 2M to upload_max_filesize = 24M
```

Now, save the changes, and when back at the command prompt, issue the following command:

```
nagitest:/etc dean$ sudo apachectl restart
```

Restarting the Apache web server forces the PHP library to (re)read its configuration file. At the bottom of the SquirrelMail page, the new max upload size is now reflected—24 megabytes.

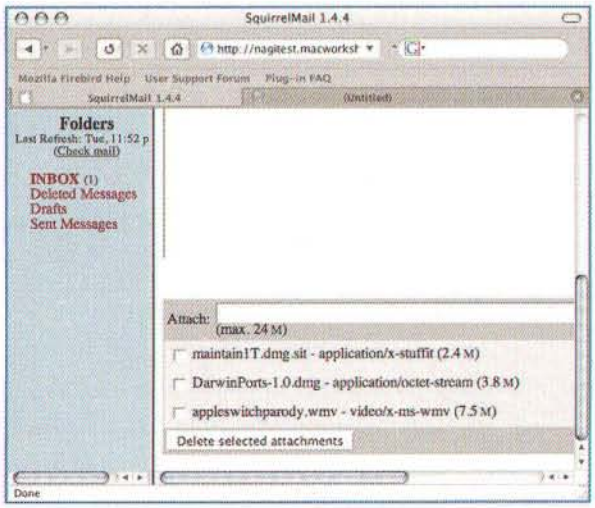


Figure 9. Souped up nuts! SquirrelMail stuffs its cheeks with attachments

Necessary Adjest[sic]ment, Fixing the SpellChecker

By default, the SquirrelSpell plugin of SquirrelMail is disabled in OS X Server, because turning it on (easy to do in the plugins section of the configure script) reveals that the ispell library it's looking for is nowhere to be found on OS X. And while there might be a nice dictionary application and widget built into OS X 10.4, enabling the spellchecker in Mac OS X Server webmail is the one of the most requested enhancements, second only to increased attachment sizes.

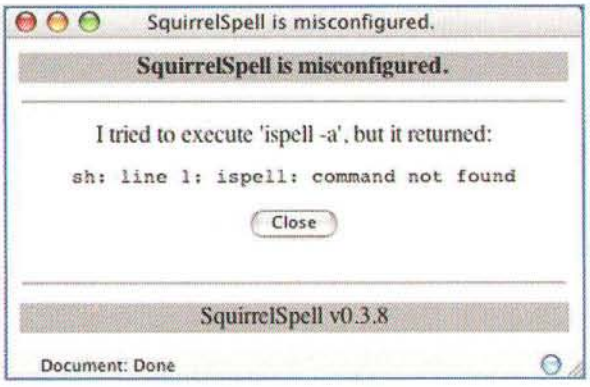


Figure 10. Where the #%!@#!@# is ispell?

The SquirrelSpell plugin expects to find ispell in /sbin but it's not included with the base install. OS X Server webmail is strictly BYOSC (bring your own spellchecker). There's three ways to get a working copy of ispell. One is to download the source code, patch it by editing one of the header files, and then install it. That, however, is just a bit too much work for something this straightforward. The other two methods are either to use the Fink package manager http://Fink.sourceforge.net, which I detailed in my February Source Hound Column, or the DarwinPorts package manager, which I'll use for this example. In using either of the three methods, an installation of Xcode 2.0 is required if installing onto OS X Server 10.4.

First, obtain and install either Xcode 1.5 if using Panther Server, or Xcode 2.0 if using Tiger Server. Next, download a copy of the Darwin ports binary installer from darwinports.opendarwin.org/downloads/DarwinPorts-1.0.dmg. Install the DarwinPorts software.

Next, navigate to /opt/local/bin, and issue the following command:

```
nagitest:/opt/local/bin dean$ sudo ./port install ispell
```

If all goes well, DarwinPorts will download, configure, compile, and even clean up after the ispell installation, which will resulting in a compiled ispell at /opt/local/bin/ispell. Now all that's necessary is to fool the SquirrelSpell plugin into thinking that ispell is in /sbin, where it's supposed to be, and that's easy enough to accomplish with a symbolic link:

```
nagitest:/opt/local/bin dean$ sudo ln -s
/opt/local/bin/ispell /sbin/ispell
```

Now, the SquirrelSpell plugin will be happy, and the spellcheck will work as advertised, making the end users smile and chant with glee "thank you SquirrelMail, thank you!" But of course, they'll be thanking no one, not even their system administrator, because it should have worked, right out-of-the-box, everyone knows that all good email clients have spellcheckers, right?

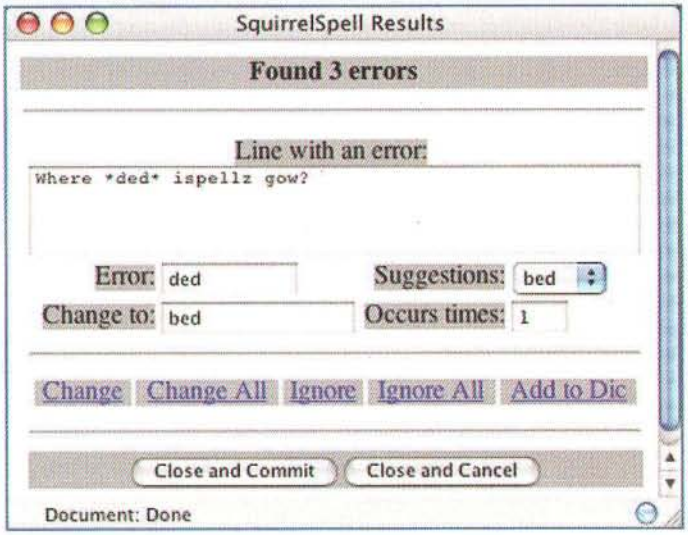


Figure 11. Where's ispell? There's ispell!

Take the Squirrel for a Walk

So now that SquirrelMail's using secure authentication, can handle decent-sized attachments, and can catch the horrid spelling errors we all make while typing, is this the end for our beloved flying squirrel, or can we take it places it hasn't been before? Well, SquirrelSpell's just the beginning. Visit SquirrelMail.sourceforge.net and check out some of the myriad

of plugins, everything from server-side plugins for spam filtering to out-of-office autoresponders to LDAP integration plugins for the SquirrelMail address book are available to try, not to mention some fun stuff like appearance themes. SquirrelMail might be inherently simple, but it has plugin support that no other webmail program can touch. What else could be more uplifting and useful than a Squirrel that makes email fly away from the office, what would be its perfect companion?

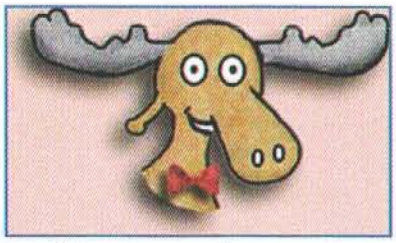


Figure 12. A Talking Moose, of Course: "A problem drinker is one who never buys."

http://www.zathras.de/angelweb/moose.htm

In Next Month's Source Hound

I will return to the subject of my very first article for MacTech back in November of last year—Apple's Directory Services a.k.a "Open Directory" I'll take a look at version III and the new schema additions, features, and tools included with Tiger Server, as well as some open-source goodies like phpLDAPadmin and ask a Tiger Open Directory Master to say "Ahhhhh" and take a deep look inside...maybe for an OU or two?

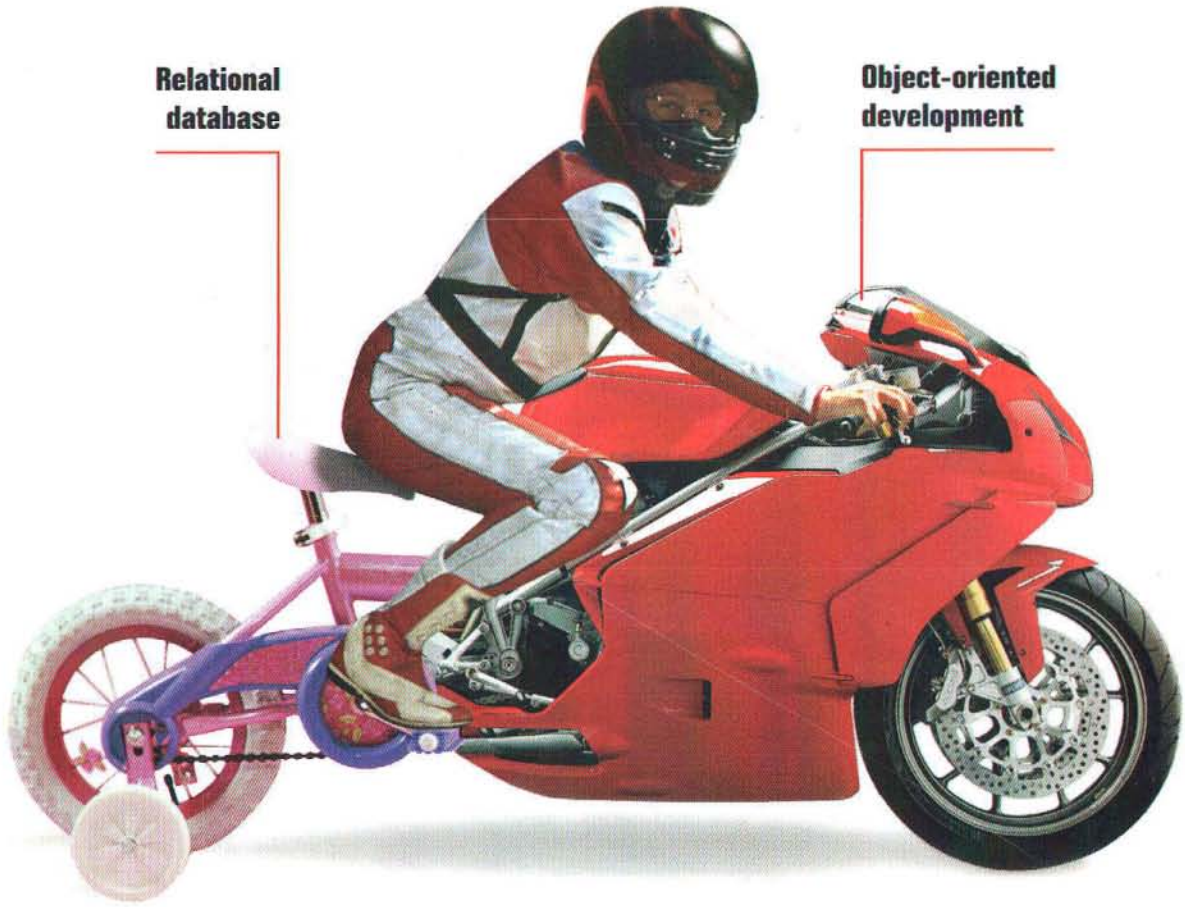
MT

About The Author

Dean Shavit is an ACSA (Apple Certified System Administrator) who loves to use a Mac, but hates paying for software. So each month he's on the hunt for the best Open-Source and freeware solutions for OS X. Besides surfing for hours, following the scent of great source code, he's a partner at MOST Training & Consulting in Chicago, where he trains system administrators in OS X and OS X Server, facilitates Mac upgrade projects for customers, and writes for his own website, www.themachelpdesk.com. Recently, he became the surprised father of an application: Mac HelpMate, available at www.machelpmate.com. If you have questions or comments you can contact him: dean@macworkshops.com.

Advertiser/Product Index

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.



A BETTER DATABASE CAN SPEED UP YOUR DEVELOPMENT CYCLE

If your relational database isn't a good match for your object-oriented development, you need a new database.

Caché, the *post-relational* database from InterSystems, combines high-performance SQL for faster queries and an advanced object database for rapidly storing and accessing objects. With Caché, no mapping is required between object and relational views of data. That means huge savings in both development and processing time.

Applications built on Caché are massively scalable and lightning fast. They require little or no database administration.

More than just a database system, Caché incorporates a powerful Web application development environment that dramatically reduces the time to build and modify applications.

Caché is so reliable, it's the world's leading database in healthcare – and it powers enterprise applications in financial services, government and many other sectors. With its high reliability, high performance and low maintenance, Caché delivers your vision of a better database.

We are InterSystems, a specialist in data management technology for over twenty-six years. We provide 24x7 support to four million users in 88 countries. Caché is available for Windows, OpenVMS, MAC OS X, Linux, and major UNIX platforms – and it is deployed on systems ranging from two to over 50,000 simultaneous users.

InterSystems
CACHÉ
Make Applications Faster

Try a better database. For free.

Download a free, fully functional, non-expiring copy of Caché or request it on CD at www.InterSystems.com/match3

The new **HASP** family of products is the next generation in protection ensuring the highest level of security for your software. It provides an easy to use set of tools to protect once and deliver through a variety of licensing options.

1 Develop

- For Windows, Mac OS, Linux

2 Protect Once

- Data encryption with industry standard hardware-based AES 128-bit
- Strong wrapper with code obfuscation prevents debugging and reverse engineering
- API code generator for easy implementation and customization

3 Distribute Many

- Single and multi-user license management
- Secure license updates for keys in the field
- Innovative licensing models defined separately from protection

Get the FREE kit: Developer's Software, Guide and Demo Key at www.aladdin.com/hasp

Aladdin
SECURING THE GLOBAL VILLAGE

North America: 1-800-562-2543, 847-818-3800 • **UK** • **Germany** • **Israel** • **Benelux** • **France** • **Spain** • **Asia Pacific** • **Japan**